Abstract

Traditionally, grasp and arm motion planning are considered as separate tasks. This might lead to problems such as limitations in the grasp possibilities, or unnecessary long times in the solution of the planning problem. This thesis presents an integrated approach that only requires the initial configuration of the robotic arm and the pose of the target object to simultaneously plan a good hand pose and arm trajectory to grasp the object.

The planner exploits the concept of independent contact regions to look for the best possible grasp. In this document, two different methods have been considered to search for good end-effector poses. One biases a sampling approach towards favorable regions using principal component analysis, and the other one considers the capabilities of the robotic arm to decide the most promising hand poses. The performance of the methods is evaluated using different scenarios for the humanoid robot SpaceJustin. In order to validate the paths, some scenarios were replicated in the laboratory and the generated paths were executed on the real robot.

Contents

List of Figures

List of Tables

Nomenclature

CMU Carnegie Mellon University

DLR Deutsches Zentrum für Luft und Raumfahrt, German Aerospace Center

FC Force Closure

FK Forward Kinematics

GWS Grasp Wrench Space

ICRs Independent Contact Regions

IK Inverse Kinematics

OMPL Open Motion Planning Library

OpenRAVE Open Robotics Automation Virtual Environment

PCA Principal Component Analysis

PRM Probabilistic Roadmap

RRT Rapidly exploring random tree

TCP Tool center point

UPC Universitat Politècnica de Catalunya

Preface

This thesis has been developed at the German Aerospace Center (DLR) in the Institute for Robotics and Mechatronics. During the thesis the author has been part of the "Robotic Grasping and Manipulation" team of the mentioned institution, having the chance to work and collaborate with many researchers on the field of grasping for robotics applications.

The main result of the project is a software module that can be easily integrated into other software packages. For this thesis, the module has been successfully integrated into The Kautham Project (a motion planner developed at the UPC) as well as with OpenRAVE (Open Robotics Automation Virtual Environment, developed at CMU).

The results of the work have been published in the paper "Integrated Grasp and Motion Planning using Independent Contact Regions," presented at the IEEE-RAS International Conference on Humanoid Robots, November 18-20th 2014, in Madrid, Spain.

Chapter 1

Introduction

Grasp and motion planning for robotic arms are a set of techniques focused on deciding how an object should be grasped and how the robot reaches the position where the grasp is executed, respectively. They have been traditionally considered as two separated stages, solved in a sequential manner to find a valid trajectory that allows a robot to pick up an object from the environment. This thesis proposes a novel method to integrate grasp and motion planning into a unified planning algorithm that does not restrict the possible grasps on the object, while at the same time computes a collision-free path plan to reach and grasp the object.

The project proposes a new approach that looks for a feasible grasp on the object exploiting the concept of independent contact regions (ICRs). ICRs have been successfully used as part of a shared autonomy approach for telemanipulation, where the human decides the hand pose with respect to the object based on the ICR information [4, 5]. This work uses the ICRs as part of a fully autonomous motion planner. In order to replace the human input, and to improve the efficiency of the planning process, the search of a new grasp is restricted to areas of potentially good configurations using two tools: 1) A principal component analysis (PCA) that biases a sampling-based search of hand poses towards promising regions, and 2) an efficient representation of the reachability of the robot in a capability map [6], which restricts the search to feasible manipulation areas for the robot. A great exploration capacity is required to simultaneously combine grasp and motion planning, therefore

bidirectional RRTs are used for the planning strategy.

After this introduction, the thesis is developed in the following way. Chapter 2 presents the state of the art in methods for motion and grasp planning. Chapter 3 describes the hardware and software tools required for the development of the proposed method. Chapter 4 presents in detail the proposed method and its algorithms. Then, Chapter 5 discusses the implementation of the method. Results are shown and analyzed in Chapter 6. The required environmental analysis is presented in Chapter 7 and the costs analysis in Chapter 8. Finally, conclusions and future work based on the results obtained in this thesis are presented.

Chapter 2

Related Work

2.1 Path Planning

Path planning (also known as motion planning) defines the process of finding a valid trajectory between an initial and a final configuration, while ensuring that all the possible constrains are respected during the execution of the complete movement. These constraints can be imposed by the environment or the kinematics and dynamics of the system; typical restrictions include torque and joint limits, obstacle avoidance or torque minimization. Motion planning is not only used in robotics, but also in other fields such as biology or digital animation [7]. Paths are computed inside the Configuration Space C ($C-space$ or also joint space) of the system. C describes the space of all possible robot configurations. For a floating base robot with N joints, a general topology is $SE(3) \times \mathbf{R}^N$. For a fixed base robot, $SE(3)$ is not present.

The workspace of a robot represents the portion of the space that the end effector of the robot can access. When orientations and positions are considered, it is a portion of $SE(3)$. The correspondence between the configuration space and the workspace of a robot is obtained via Forward and Inverse Kinematics (FK and IK). Forward Kinematics uses the kinematic equations of the robot to compute the end-effector position and orientation. On the other hand, Inverse Kinematics tries to

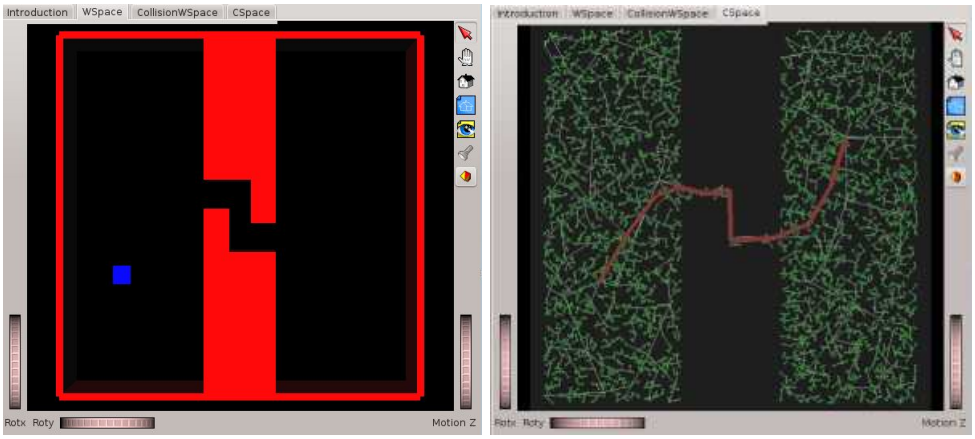

Figure 2.1: Typical path planning problem. Taken from The Kautham Project [1].

find which configuration brings the end effector to a desired position. The problem of IK is that there is not always an analytical way to solve the problem, since many manipulators are redundant, i.e., have more than six joints and have multiple configurations that can reach the same end effector pose. In these cases, IK is normally solved as a optimization problem. Another way to tackle this problem is using the pseudo-inverse of the Jacobian to find correspondences between small changes in the workspace with small changes in the configuration space [8].

The problem of collision-free arm motion planning, i.e. finding an arm trajectory from an initial to a final configuration while avoiding obstacles in the environment, has been tackled using mainly two approaches: sampling- and optimization-based planning. Because of their simplicity, sampling-based motion planners have become very popular. They approximate the configuration space by sampling it (uniformly or not) and keeping the collision-free samples in a graph structure defining the free configuration space (C_{free}). Sampling-based planners include methods based on probabilistic road-maps (PRM), where several collision-free configurations are computed beforehand and stored in a graph structure called roadmap [7]. In this graph, the nodes represent configuration samples while the edges represent local paths connecting nodes that are close enough to each other in the configuration space. The roadmap is later queried to get the desired path by checking the connectivity between two nodes in the graph. The approach is particularly useful for multiple queries in

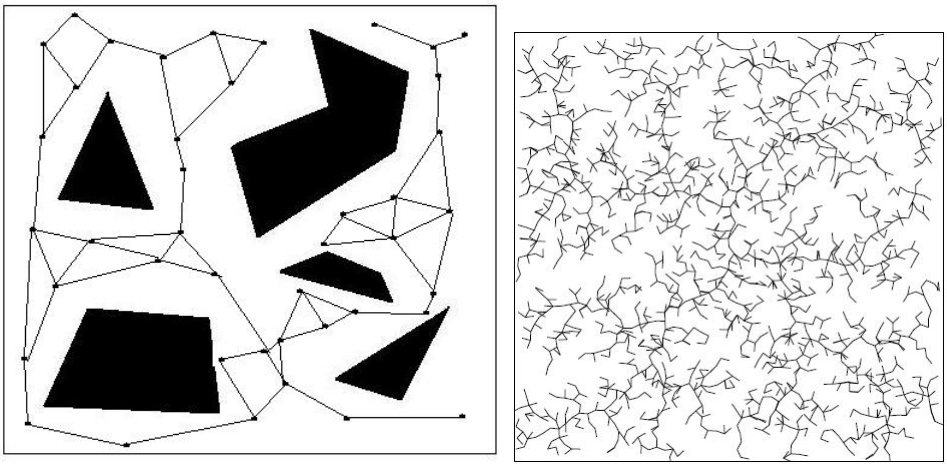

Figure 2.2: Examples of sampling-based path planners. Left: PRM. Right: RRT.

very structured and static scenarios. For single queries, rapidly-random exploring trees (RRTs) sample the configuration space while creating a tree that connects the start and goal configuration [9], which proves useful for changing environments like those encountered in manipulation tasks with different obstacles.

These randomized path planners are probabilistically complete, meaning that as the number of steps executed from the planner increases, the probability that the planner fails to find a path, if one exists, asymptotically approaches zero. One disadvantage of these planners is that they often return non-optimal solutions that require a post-processing step to smooth and shorten the computed trajectories. Examples of 2D planning using these techniques are sketched in Fig. 2.2.

Optimization-based planners, on the other hand, define path computation as an optimization problem that minimizes a suitable cost function [10]. Such cost function can be used to avoid obstacles, provide smoothness, and respect kinematic and dynamic constraints of the problem. These planning approaches suffer from limitations associated to optimization algorithms such as non-convergence or local minima, and lack the exploration nature of sampling-based planners required for difficult path planning problems. The consideration of optimality within sampling-based planners has been recently proposed [11], although these approaches are not computationally efficient for high-dimensional problems.

2.2 Grasp Planning

Grasp Planning involves a set of techniques whose goal is finding good contact locations on the object and a corresponding suitable hand configuration, given a particular end effector (gripper or multi-fingered hand) and object. The grasp is usually defined as a homogeneous transformation that defines the relative location of the object with respect to the gripper, and the set of joint values that move the gripper to the desired contact points on the object.

Depending on the way that the grasp configurations are searched, grasp planning approaches can be separated into analytical and data-driven methods [12]. Typically, analytical methods tackle the grasp search with heuristic- or optimization-based algorithms, adapting the restrictions and goal functions to the particularities of the problem. On the other hand, data-driven methods mainly rely on an off-line generated set of grasp configurations, which is later used to choose a valid grasp pose according to other restrictions of the environment. Grasps are chosen according to some desired property, being force closure (FC) the most popular one. A grasp that is FC is able to resist perturbations in any direction, using suitable contact forces. To measure the goodness of a grasp, different grasp quality measures have been proposed [13]; the most common one is the measure of the maximum perturbation wrench that a grasp can resist in any direction [14].

Most of the approaches for planning grasps provide precise contact locations for the fingers on the object surface, but real robotic hands can hardly guarantee that the exact contact points are reached due to different sources of uncertainty (for instance, a poorly estimated friction coefficient, or uncertainties in the object pose estimation). The computation of independent contact regions (ICRs) on the object boundary was introduced to provide robustness to finger positioning errors, such that if each finger i is positioned on its corresponding ICR_i an FC grasp is always obtained, independently of the exact location of each finger [15].

2.3 Integrated grasp and motion planning

Finding a feasible grasp for a given scenario has been traditionally solved in a sequential manner. Given the object model, a grasp database is computed offline, and grasps are sorted according to some grasp quality measure. Later, given the current scenario, the feasibility of the grasps is evaluated, i.e. only grasps that have a corresponding inverse kinematics (IK) solution for the arm are considered in later stages. One feasible grasp defines one goal configuration for the robot. Then, given the initial and final arm configuration, a collision-free path for the arm is searched using some path planning method. If no path is found a new grasp is chosen, until a path is obtained or until the complete database has been explored and no solution is found [16]. Working with a discrete set of grasps from a database has several disadvantages, such as limitations in the number of grasp possibilities for the object and low adaptability to the environment, i.e. no new grasps can be explored even if they mean only a slight change of pose with respect to one predefined grasp.

While traditional motion planning for grasping connects an initial to a desired arm/hand configuration, an integrated approach simultaneously looks for the hand configuration and arm motion to grasp a particular object given the initial arm/-hand configuration and object pose. In most of the cases it is not important how the object is grasped, as long as some minimum quality conditions are met.

An initial step towards an integrated approach was taken by defining online a set of candidate grasp poses, obtained by solving an optimization problem that takes into account the location of obstacles in the scene and the likelihood of not being in collision and leading to FC grasps [17]. Candidate grasps obtained in this way are then tried out by moving the wrist and closing the fingers until some grasp is detected. This idea evolved towards the definition of Task Space Regions [18, 19], which manually define areas (subsets of $SE(3)$) of predefined good grasp poses around the object for an underactuated hand. The regions are later used to plan arm paths with a bidirectional RRT.

The previous work closest to the approach presented in this thesis is [20], where the

grasp search is performed while the motion planning loop is running. From the initial arm configuration, an RRT is built and starts growing. Occasionally, an approach movement is tried from some existing node in the tree towards the object, stopping as close as possible to the object while still there are no collisions detected. Once the hand is at this position, a grasp is evaluated by simply closing the fingers around the object. During the whole planning process the approach trajectories towards the object are tracked, to try to uniformly cover different approach directions.

This thesis proposes a novel way to integrate the grasp and motion planning procedures without using any precomputed grasp database, thus making it suitable for arbitrary objects and complex end effectors. The approach reuses the idea of a goal region around the object that contains promising hand poses, but the proposed approach does not explicitly constrain the potential grasps on the object. The method also provides robustness against uncertainty in the positioning of the fingers by computing reachable independent contact regions, which also leads to more candidate grasps than the closing and testing policy adopted in some common grasp planning approaches. To improve the performance in cluttered environments, bidirectional RRTs are used to connect the initial configuration with the multiple grasps discovered during the search process. The planning algorithm tries to bias the connection of the trees to the ones leading to better grasps, measured according to a suitable quality metric.

Chapter 3

Tools

3.1 Robot SpaceJustin

Although the integrated planning method that was developed and will be presented in Chapter 4 can be used with any robotic arm and any manipulator, the examples and real implementation was done for SpaceJustin (Fig. 3.1), the upper body of a humanoid robot developed by the German Aerospace Center (DLR) at the Institute of Robotics and Mechatronics in Oberpfaffenhofen. The main purpose of SpaceJustin is to work in a telepresence environment, for instance for teleoperation from the earth for tasks on the International Space Station.

The robot SpaceJustin has two arms of seven degrees of freedom (DoF) each, plus one neck with 2 DoF and one additional DoF in the waist; each drive has position and joint torque sensors. The robot uses as end effectors two five-fingered DLR-HIT hands II with 15 DoF each. These are anthropomorphic hands also designed by the German Aerospace Center, which are commercially available. The hand has 5 fingers with 4 joints each of them, although the two last joints are coupled with a relation 1/1, thus leaving 3 controlled degrees of freedom per finger. Every joint is also equipped with torque sensors. Given the small workspace obtained by the original hand configuration shown in Figure 3.2, the configuration of the thumb was changed. In the new configuration the thumb is placed totally opposed to the index

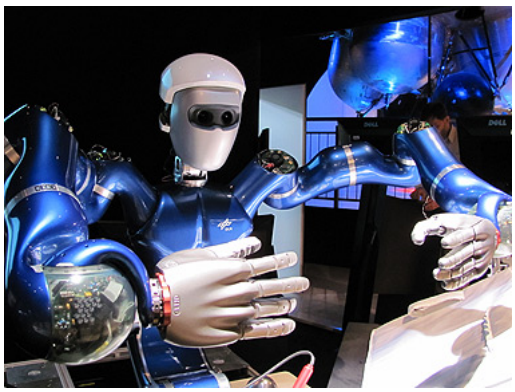

Figure 3.1: Humanoid robot SpaceJustin.

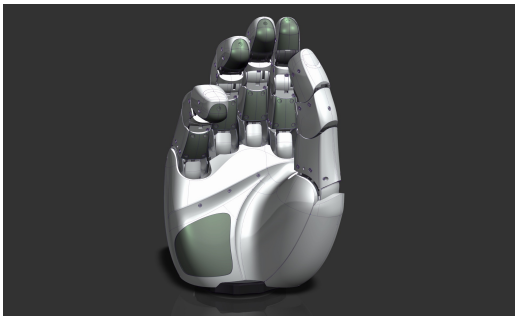

Figure 3.2: DLR-HIT hand II in its original configuration.

finger. This new configuration does not just give the hand a larger workspace, but it also provides an important increment of dexterity [21].

3.2 Independent Contact Regions (ICR)

One of the most popular methods in the evaluation of grasp robustness is the force closure (FC) method, generally used with the additional consideration of a grasp quality measure. They are based in the analysis of the Grasp Wrench Space (GWS). The Grasp Wrench Space is a subset of the 6-dimensional force and torque space spanned by the forces applied on the contact points of the object and the torques produced by those forces, measured with respect to some convenient reference point, usually the center of mass of the object. These forces are the forces that each contact can apply at the contact point, considering the restrictions coming from the friction

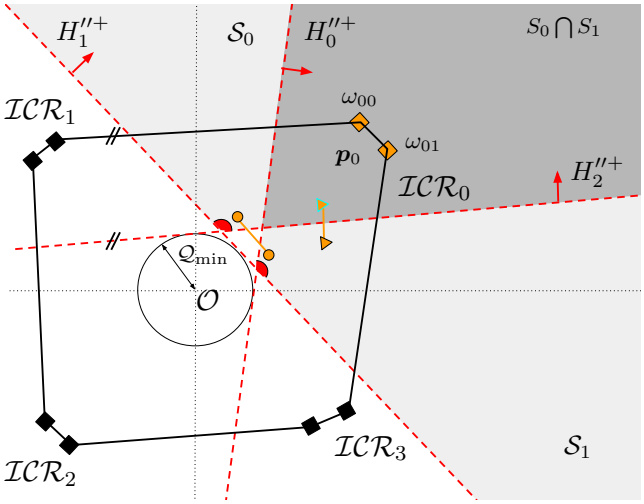

Figure 3.3: ICR computation in an abstract 2-dimensional wrench space (taken from [2]).

cone. A convex hull of all these points in the GWS is computed and from that the FC test is evaluated by checking if the origin of the wrench space lies inside the convex hull. If the test turns positive it means that the object grasped can resist perturbations in any possible direction.

To measure the quality for different FC grasps, a metric is required to give an idea of the disturbance resistance of a grasp. One of the most popular metrics is the largest ball criterion, which computes the minimum distance in the wrench space from the origin to any of the facets forming the convex hull. This quality metric also defines the direction of the maximum perturbation wrench that the object can handle without loosing the force closure condition.

ICRs correspond to regions on the boundary of the object (here represented by sets of points), such that if each finger i is positioned on its corresponding ICR_i an FC grasp is always obtained. Not only the FC condition is guaranteed, but also a minimum desired grasp quality is met. The grasp quality is quantified with the largest perturbation wrench that the grasp can resist in any perturbation direction [14]. The computation of independent contact regions was introduced to provide robust-

ness against finger positioning errors. For its computation, the contact between the finger and the object surface are considered punctual. A force f_i applied at a point p_i generates a torque $\tau_i = p_i \times f_i$ with respect to an object reference. This reference frame of the system is located in the center of mass CM for simplicity. f_i and τ_i form together a wrench vector $\omega_i = (f_i\ \tau_i)^T$. For the ICR computation, the friction caused by the contact between the fingertips and the object is described by a friction cone. To simplify the computations, the friction cone is linearized using an m-side polyhedral convex cone. The number of sides considered for that cone is the result of a compromise between the desired precision and the computational time, since the higher this number is the higher the computational time. A wrench ω_{ij} generated by a unitary force f_i along an edge of the linearized friction cone, i.e. $f_i = \hat{n}_{ij}$, is called a primitive wrench.

Each contact point p_i has m corresponding primitive wrenches ω_{ij}. A grasp is defined by the set of contact points $\mathcal{G} = \{p_1, \ldots, p_n\}$, and thus associated to the set $\mathcal{W} = \{\omega_{11}, \ldots, \omega_{1m} \ldots, \omega_{n1}, \ldots, \omega_{nm}\}$. Therefore, $\mathcal{G}$ and $\mathcal{W}$ will be used as representative sets of a particular grasp. It is important to notice that, unlike the definition proposed in Chapter 2.2, in this grasp definition the transformation of the gripper is not considered. This is because the Independent Contact Regions algorithm does not explicitly take it into consideration, but it is a previous step as will be described below.

Initially, the FC condition is tested. First, the convex hull of the grasp $CH(\mathcal{W})$ is computed for all the primitive wrenches of the contact points. Next, the condition that the origin is found inside $CH(\mathcal{W})$ is evaluated by verifying that for every facet of the convex hull, the centroid P of all the primitive wrenches lies on the same side as the origin (the center of mass) [22] This condition is valid since the centroid will always lie inside the $CH(\mathcal{W})$. The quality of the grasp is the radius of the largest hypersphere centered on O and fully contained in $CH(\mathcal{W})$, i.e. it is the distance from O to the closest facet of $CH(\mathcal{W})$.

The computation of the Independent Contact Region starts when a first FC candidate grasp is found. The method generates hyperplanes H_k'' parallel to the support-

ing hyperplanes H_k, i.e. planes that contain one facet of $CH(\mathcal{W})$. Each hyperplane generates a positive and negative half-space. Arbitrarily, the negative half space is selected to be the one where the origin O of the wrench space lies. A search region S_k is defined as the positive half space $H_k''^{+}$, meaning that new points in the $\mathcal{ICR}_i$ will have its primitive wrenches in this region. A simple search is performed to build each $\mathcal{ICR}_i$ by looking for the neighbor points of p_i such that at least one of its primitive wrenches lies on each search region S_k. The points that fulfill this condition belong to the region $\mathcal{ICR}_i$. This search is performed for all the fingers involved.

The reachability of the points computed by the ICRs is considered in a two-phases approach [23]. The first one obtains the points on the object surface reachable by the fingers for a given hand pose; points are represented by a position vector and a corresponding normal direction. Precomputation of the workspaces for each finger speeds up this process. The second phase executes the ICR algorithm previously described, but only using reachable points, which therefore leads to reachable independent contact regions.

In order to find the reachable ICRs the first step required is to find those points on the surface of the object which are actually reachable for every finger given a position of the hand. Thus, a voxelized representation of the finger's workspace is used. The fingers used during the development of the method just have 3 DoF, since the two last degrees of freedom are coupled. This eases this representation because the mapping from the workspace to the configuration of these 3 DoF is unique, meaning that one position in this representation has only one possible orientation of the fingertip and just one element of the subset $\mathbf{R}^3$. Every voxel in the workspace holds then information about the position of the fingertip with respect to the finger origin, a vector representing the ideal direction where the finger tip can actually apply force, and a 3-dimensional vector indicating the finger configuration that leads to that position. With these properties, this representation serves also as an inverse kinematics look-up table.

The reachable points of the object are computed by the collision detection between

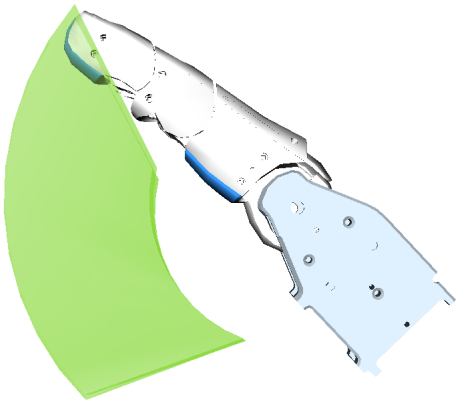

Figure 3.4: Workspace for one finger of the DLR-HIT hand II.

the worskpaces and the object. This collision detection is performed by a checker that queries the collision between a voxelized map and a point cloud that in this case represents the object. The technique used is called VPS, or Voxmap-Pointshell Method [30]. It does not just give information about whether a collision takes place or not, but it also provides the point on the point cloud and the voxels that are colliding. But not all the points that this method gives are useful to execute valid grasps, since in many cases the fingers cannot apply force in the normal direction of the object's surface. Instead, every point is checked with its corresponding voxel to check if the normal vector of the object's surface and the vector associated to the voxel form an angle smaller than 45 degrees 3.5. Once these points are found, the ICR algorithm is applied as explained before.

For the integrated planner, an additional quality metric for the ICRs is required. As the regions are represented as discrete sets of points, the more points in the region the more possibilities of grasping the object are provided. The metric is then defined as the number of different grasp possibilities, which is indirectly related to the area that the regions cover on the object surface. Let n be the number of fingers with contact regions on the object surface, and m_i be the number of discrete points in

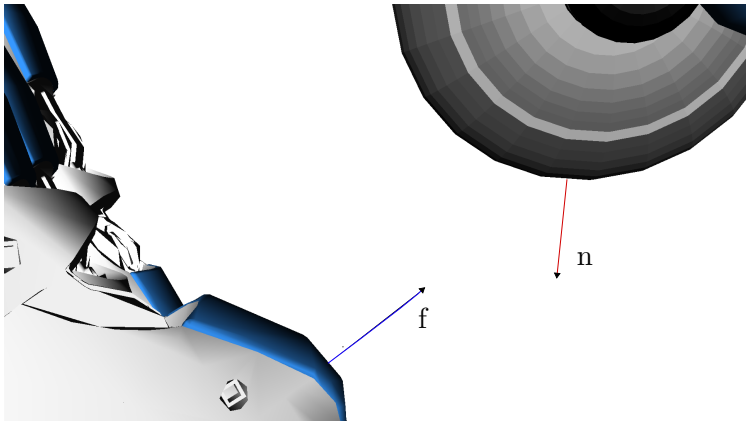

Figure 3.5: Finger workspace collision detection. n is a normal vector on the object's surface and f the vector representing the ideal direction where the finger tip can actually apply force. The angle between f and $-n$ is checked for reachability.

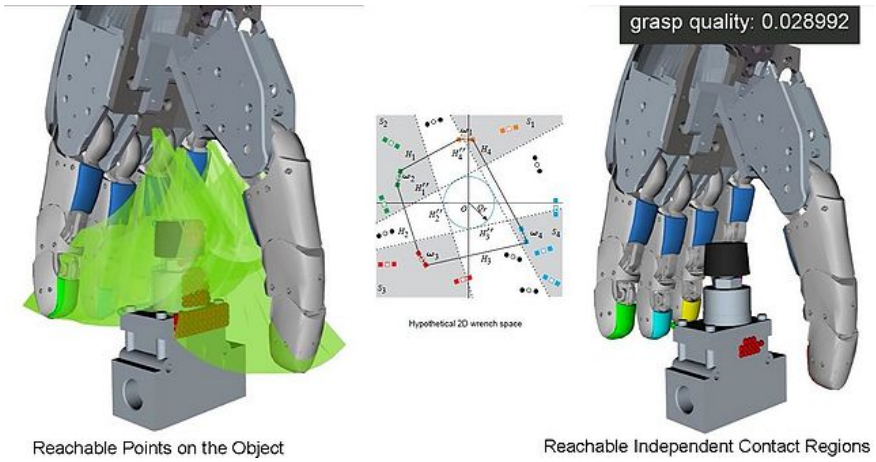

Figure 3.6: Visual representation of the complete ICR computation process.

each region ICR_i; the quality metric for the contact regions for a particular hand pose is given by

$$Q_{ICR} = \prod_{i=1}^{n} m_i \tag{3.1}$$

This quality will serve as weight factor that the planner will use to decide which grasp is better to perform. The highest value Q_{ICR} reaches, the more robust the grasp can be against uncertainties in finger and object positioning errors.

3.3 Capability Map

In general, an offline analysis of the reachability of a robotic arm saves time for online queries like grasp selection or path planning. The representation of the regions where the robot tool frame (TCP) can be moved to is known as a reachability map [24]. The reachability data is defined by indices quantifying how "good" a region of the space is. Arms with different kinematical structures provide different capabilities. Either if the mobile robot is located in a suitable position or if the robot arm has a fixed base, an offline analysis of the robot's workspace is helpful to speed up the online solution of planning tasks.

The reachability map is in general computed as a spatial grid in the 6D space (position and orientation), where each cell has a binary value that indicates if it is reachable or not, together with an associated quality index measuring the dexterity of the robot when the TCP is at that position. A high dexterity is considered in the sense that the number of different orientations that the end effector can reach while staying at the same position is high. The information of the reachability map requires an accurate description of the workspace, usually described with a voxelized structure. The reachability or capability maps can be used for online computation and selection of reachable grasps for performing manipulation tasks. Given a grasp

pose, the position of the TCP is checked against the capability map to verify if it is reachable or not.

The offline computation of the map can be done by forward or inverse kinematics. With forward kinematics (FK), the different arm configurations are randomly sampled in joint space, the TCP is computed and then assigned to one bin in the map, which changes value from 0 to 1. This method, however, has the problems associated to redundant manipulators like the presence of singularities. Due to the singularities, a uniform sampling of the possible joint configurations does not guarantee a uniform description of the robot's workspace. Samples tend to accumulate in small regions of the space close to these singularities, thus not guaranteeing the completeness of the map. With inverse kinematics (IK), the 6D space is uniformly sampled, and for each sample an IK solver runs to try to find a solution that leads to that configuration. This method guarantees a complete exploration of the workspace, but the use of an IK solver makes the generation of a new sample much slower. Therefore, the capability map used in this project is computed offline using an hybrid approach that combines forward and inverse kinematics, to obtain an accurate and structured description of the robot capabilities [6].

The reachability map is then represented using a voxelized structure in $SE(3)$, which can be interpreted as the voxelized reachable workspace in $\mathbf{R}^3$ where each cartesian voxel has an associated rotational grid that discretizes $SO(3)$. The 3-dimensional Euclidean space $\mathbf{R}^3$ is divided into cubes $C_{x,y,z}$ of a user-defined side length. This allows the mapping of the TCP's translational component to a cubical volume, $t_{TCP} \rightarrow C_{x,y,z}$ with $x, y, z \in \mathbf{N}$.

Each cartesian voxel holds a rotational occupancy grid to map the orientation of the end effector R_{TCP}. To obtain this grid, a virtual sphere is associated to each voxel; the spherical surface represents the discretization of the approach direction (pitch, yaw of the TCP), i.e. it is a division of $SO(2)$. Each surface division of the virtual sphere has an attached discretization of the missing parameter (roll) that completes the $SO(3)$ division. Such organized division allows creating a precomputed table of

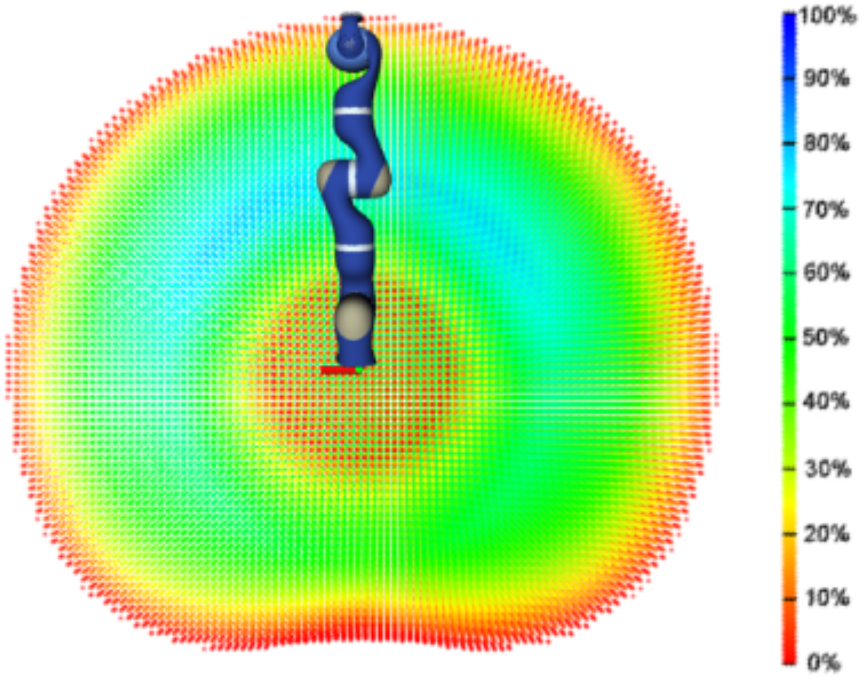

Figure 3.7: Capability map for one arm of SpaceJustin. Colors indicate the reachability index for each voxel, i.e. the percentage of discrete poses within that voxel that are reachable for the arm.

positions, which makes it fast to determine the bin that corresponds to a particular R_{TCP}. With this structure, each R_{TCP} is uniquely mapped to one of the bins.

When a $SE(3)$ configuration is queried with the capability map, the translational part is associated to the closest discrete one found into the map, and its orientation reachability is validated by the closest division in the rotational grid. Thus, the performance obtained by its use is highly dependent on the division size of the workspace. The resolution is chosen by trying to find an equilibrium between the precision and the memory size required to store the map for future queries.

Chapter 4

Integrated method

4.1 Grasp Planning

To tackle the problem of the integrated planning, two approaches have been developed in order to find valid grasps and successfully integrate them with the motion planning algorithm. The first one proposes the definition of a region around the object with the use of superellipsoids, where a principal component analysis algorithm is applied to try to guide and speed up the sampling of valid positions around the object in the search for feasible grasps. The second one proposes the use of the capability map to filter the area around the object which is potentially valid to perform valid grasps.The two methods are described in the next sections and are both used in the fully integrated method.

4.1.1 Using PCA

During the execution of the integrated planning algorithm, one requirement is to find configurations of the hand that will likely lead to a valid grasp. To achieve this purpose, a goal region around the object is defined using a superellipsoid. Samples of potential hand positions are taken inside this space, and an adaptive sampling method based on PCA is applied to bias the sampling towards promising areas. This follows the idea proposed in [25], where the PCA guides the sampling process in path

planning problems with narrow passages by adapting the region in the configuration space from where the next sample is taken.

Superellipsoids are a way to represent any 3-D surface obtained by the cross product of two superellipses. A superellipse is defined as:

$$s(\theta) = \begin{bmatrix} a\cos^{\epsilon}\theta \\ b\sin^{\epsilon}\theta \end{bmatrix}; -\pi \leq \theta \leq \pi \tag{4.1}$$

And a superellipsoid is obtained by:

$$r(\eta,\omega) = s_1(\eta) \bigotimes s_2(\omega) \tag{4.2}$$

$$r(\eta,\omega) = \begin{bmatrix} a_1\cos^{\epsilon_1}\eta\cos^{\epsilon_2}\omega \\ a_2\cos^{\epsilon_1}\eta\sin^{\epsilon_2}\omega \\ a_3\sin^{\epsilon_1}\eta \end{bmatrix}; \begin{matrix} -\pi/2 \leq \eta \leq \pi/2 \\ -\pi \leq \omega \leq \pi \end{matrix} \tag{4.3}$$

Here, the parameters a_1,a_2,a_3 are scaling factors along the three coordinate axis, while ϵ_1 and ϵ_2 are shape parameters. The shape parameters ϵ_1 and ϵ_2 must be selected such that the goal region represents the shape of the real object while avoiding the concentration of samples in undesired places (Fig. 4.1).

The goal region is then defined by the set A

$$A = \left\{ (a_1,a_2,a_3,\eta,\omega)^T \middle| \begin{matrix} a_{i_{min}} \leq a_i \leq a_{i_{max}} \\ -\pi/2 \leq \eta \leq \pi/2 \\ \omega_{min} \leq \omega \leq \omega_{max} \end{matrix} \right\} \tag{4.4}$$

where $a_{i_{min}}$ and $a_{i_{max}}$, $i = \{1,2,3\}$, define the region of interest along each coordinate axis (for instance, to avoid points below or too close to the supporting surface).

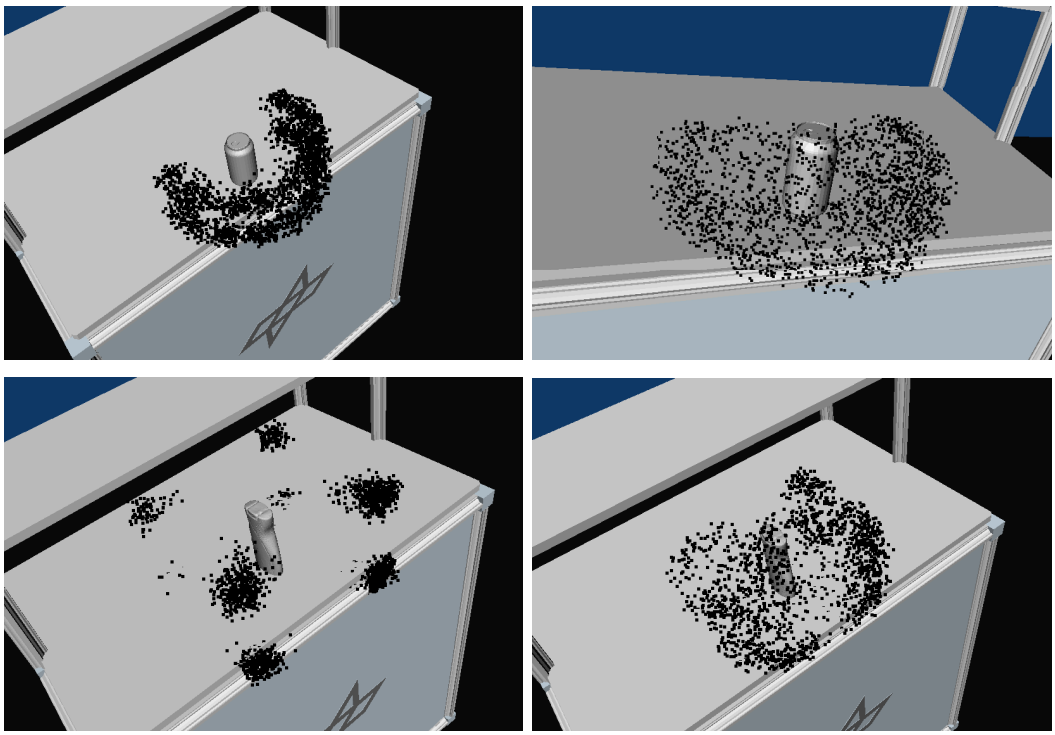

Figure 4.1: Influence of the superellipsoid parameters on the distribution of samples inside the goal region for a soda can in the top row (with $\epsilon_1 = 0.1$, $\epsilon_2 = 1.0$ to the left and $\epsilon_1 = 0.5$, $\epsilon_2 = 1.0$ to the right), and for a shampoo bottle in the bottom row (with $\epsilon_1 = 0.1$, $\epsilon_2 = 0.1$ to the left and $\epsilon_1 = 0.5$, $\epsilon_2 = 0.5$ to the right).

ω_{min} and ω_{max} choose the angular portion of the superellipsoid that faces the robot, as shown in Fig. 4.1, which discards grasp poses that might look unnatural. In this work, the parameters for A were manually set, although an automatic selection process could also be implemented.

Each sample inside the 5-dimensional space A provides a Cartesian position of the hand with respect to the object coordinate frame. The orientation of the hand with respect to the object is defined in such a way that the object is always "visible" for the palm, as suggested in [26] to generate more human-like paths. This still leaves one DoF, the roll orientation for the hand around the approach axis, which is chosen such that the palm is parallel to the main axis of inertia of the object [27]. Thus, the complete hand pose with respect to the object T_{hand}^{object} is obtained, which is later used for the computation of the ICRs.

During the complete planning procedure that will be presented in section 4.2, different samples for robot tool frame (TCP) locations are taken inside A. These samples are checked to see whether they correspond to collision-free configurations of the arm/hand, and if at least two fingers of the hand have reachable points on the object surface. If they meet these conditions, they are used inside the Principal Component Analysis algorithm.

Principal Component analysis (PCA) is a statistical technique used to process a set of vectorial samples in order to look for a new orthogonal base of the vectorial space whose axis indicate, in a decreasing order, the directions of the space with more information to discriminate the samples. In the field of motion and path planning, it is frequently used to reduce the dimension of the search space and therefore to decrease the running time.

In this work, PCA is computed through the eigenvalue decomposition of a data covariance matrix. The main idea is to iteratively get an hyper-rectangle $H \subseteq A$ from where to take samples. The expected behavior is that the probability to obtain valid samples in H increases at every iteration.

To obtain H and implement the algorithm, the data matrix $PCAmat$ is needed, which contains all the samples considered valid for the purpose, in this case, collision-free samples. It is important to notice that the number of rows in the data matrix is continuously increasing as new valid samples appear. Once the data matrix is obtained, H is defined with the following three components:

- Orientation matrix M_{5x5} formed by the eigenvectors of the covariance matrix of the $PCAmat$.
- Translation vector mv taken from the mean of the data matrix.
- Extension vector e proportional to the eigenvalues of $cov(PCAmat)$

At the beginning, and until the PCA starts to work, H is defined as H_{ini} with the next components.

- Orientation matrix $M_{5x5} = I_{5x5}$

- Translation vector $mv = (0, 0, 0, 0, 0)$
- Extension vector e is fixed off-line according to the size of the axis-aligned bounding box of the object and the hand.

In the integrated motion planner, the PCA is applied to look for a good grasp pose as follows:

1 A sample is taken inside H, thus obtaining a transformation between the hand and the object.

2 If this transformation fulfills the following three requirements, the last sample obtained is inserted in the data matrix:

a The transformation is reachable $\rightarrow$ IK solution exists.

b The configuration of the hand-arm system is collision-free.

c There are reachable points for at least two fingers on the object.

3 Recompute H for next iterations.

a Computation of the data covariance matrix $cov(PCAmat)$.

b Eigenvalues decomposition of $cov(PCAmat)$.

c Obtaining new orientation matrix M_{5x5}, translation vector mv and extension vector e.

With this approach, as the sampling process advances it is adapted to the particularities of the given environment, as well as to the possible particularities in the shape of the object. It is important to notice that the PCA method in this case does not try to apply any dimension reduction to the space, but just guides the sampling process towards promising areas. A similar approach has been used in [25] to guide the sampling process in narrow passages for motion planning.

In summary, the PCA based method changes the orientation, center and extension of the sampling hyperbox while the planning method is running, which adapts the sampling policy to the singularities of the environment. For example, if the object

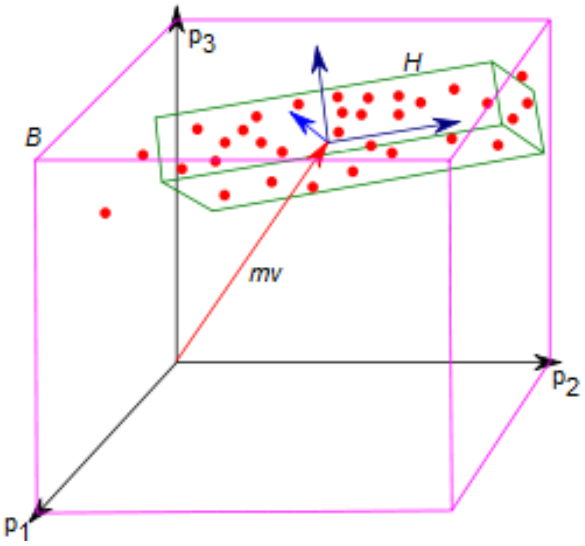

Figure 4.2: PCA-based sampling applied to a space B defined by 3 parameters; mv is the mean of the set of samples and H is the new sampling region.

that needs to be grasped is surrounded by obstacles, the algorithm will try to search grasping poses in the areas where the hand can be placed without leading to colliding configurations.

4.1.2 Using CapMap

As explained in section 3.3, the capability map provides a discretization of the workspace of the robot together with information of its reachability and dexterity. The capability map in this work has been used with two purposes: to provide a previous filtering of the regions that will likely lead to good grasps, and to query the tested grasps for reachability, thus saving time in inverse kinematics solutions.

For using the capability map, the goal region - where samples for potential hand poses are taken - is defined as a partial hollow box that surrounds the object. The outer dimension of the box is defined by the size of the hand, and the inner dimension corresponds to the length of the fingers minus the length of the object along that dimension. The goal region represented by such box is intersected with the capability map to provide the regions of the box that are reachable by the robotic arm. Note that this filtering is performed only once at the beginning of the complete planning

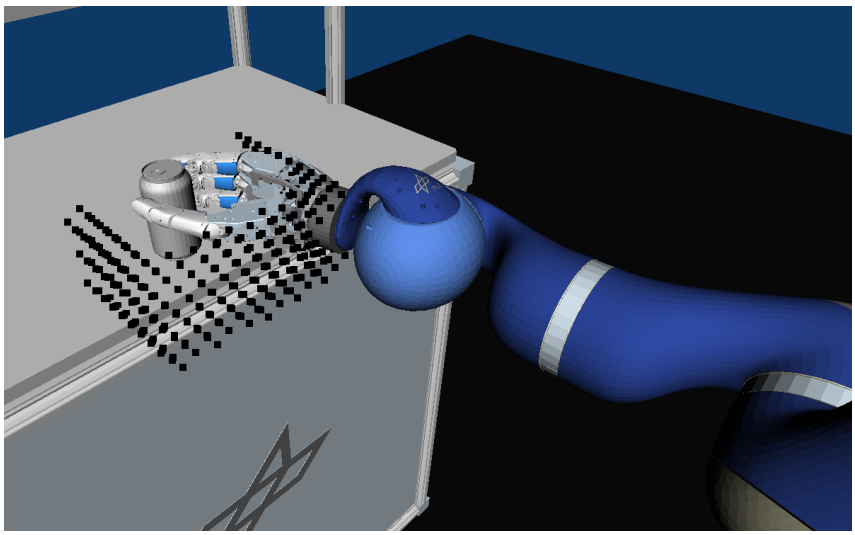

Figure 4.3: Points resulting from the filtering phase of the capability map. The points define hand positions reachable by the arm.

process, as it only needs the data of the capability map and the position of the object inside the robot workspace. The result of this filtering process is a list of positions that are potentially valid to obtain good grasps. Fig. 4.3 shows the result of this filtering step for a soda can; all the represented points are possible hand positions that are inside the goal region and are also reachable by the arm.

During the planning process, samples are taken inside this reachable region from the list of feasible positions. These samples provide the Cartesian position of the hand with respect to the object. All the points obtained from the capability map in the filtering phase have reachability in at least one direction, but it does not mean that the robot can reach one of those points while pointing towards the object in the desired way. Therefore, the computed transformation T_{hand}^{object} will be checked for reachability using the capability map, before computing the more expensive inverse kinematics.

Usage of the capability map provides two main advantages. First, it defines from the beginning a portion of the goal region reachable by the arm, thus discarding unfeasible areas where otherwise samples would have to be tossed out by calling an IK solver if the reachability information were not verified in advance. Second, it can

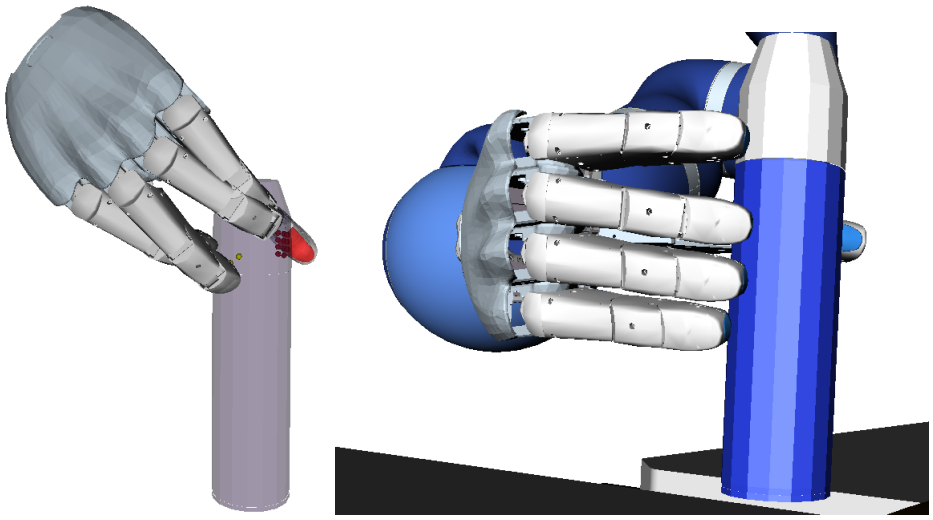

Figure 4.4: Different T_{hand}^{object} for grasping. Left: self-collision between two fingers. Right: good grasp pose.

easily incorporate additional restrictions, for instance, guaranteeing that the robot is always facing the object during the grasp process, which leads to more intuitive solutions of the planning problem.

4.1.3 Complete transformation T_{hand}^{object}

In the two grasp planning methods described before, the capability map or the PCA provide just (x, y, z) positions. To fix the orientation of the hand that should be used to perform the grasp there are still 3 degrees of freedom that need to be fixed. Two of them are fixed by defining an approach vector on the palm that points towards the center of the object [26]. With this transformation there is still one last degree of freedom that should be set, which corresponds to the rotation around the defined approach vector on the palm. This rotation does not correspond with a pure wrist rotation, since this approach vector is defined to have around 30 degrees angle with respect to the wrist rotation vector. The random sampling of this angle leads to a non-efficient way of looking for grasps, as in this case most of the grasps are unnatural and the computation of the ICR leads too often to self collision between the fingers (Figure 4.4).

The empirical solution adopted in this work is to find the rotation angle that places the x-axis of the palm lying on the plane of the object that needs to be grasped, while maintaining the approach vector pointing towards the object. The process to find this last transformation is illustrated in figure 4.5, and follows these steps

1 Find the intersection between a 60 degrees opening cone and the $x - y$ plane. This intersection defines x_{new}. This vector is the goal vector that x_{hand} has to turn to. The value of 60 degrees is taken because of the DLR-HIT hand II configuration. The 60 degrees orientation allows the palm to face the objects without risk of collision with the thumb.

2 In a plane perpendicular to the approach direction x_p, find the points that define its intersection with x_p, the axis x_{hand}, and the goal vector x_{new} corresponding to the intersection.

3 In this plane find the value of the angle that has to be rotated.

4 Get the corresponding transformation and compose it to get the final T_{hand}^{object}.

4.2 Integrated planning algorithm

This section presents the algorithm that integrates the grasp planning methods, explained in Section 4.1, with an RRT path planning approach for defining the complete grasping motion of the arm/hand system.

Given an initial pose of the object to be grasped and an initial configuration for the arm/hand system, the goal is to find a path leading to the best possible grasp, measured according to the ICR quality described in Eq. (3.1). The object is described as a pointshell, to facilitate the computation of the ICRs. Algorithm 1 formalizes the approach; particular details of the planner are presented in Algorithms 2 and 3.

The planner has two different parts that will be explained in detail below. The first one (Lines 3 to 17) looks for a valid grasp on the object, which defines a goal configuration, while at the same time grows an RRT from the initial arm/hand pose.

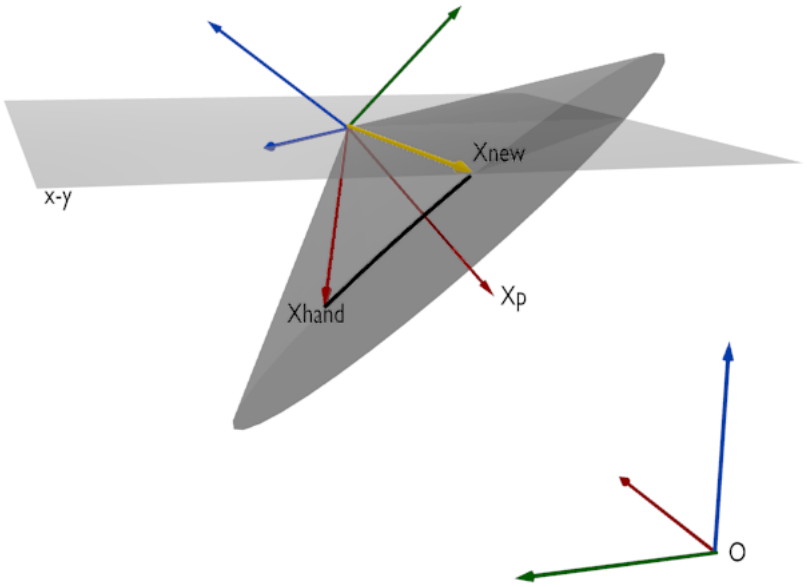

Figure 4.5: Finding the rotation around the approach vector x_p.

The second part (Lines 18 to 31) uses a bidirectional RRT to try and connect the initial configuration to the most promising grasp on the object, while still looking for higher quality grasps. The path generated by the algorithm is later smoothed using pruning techniques. Tables 4.1 and 4.2 have a list of all the data types and the methods used in the algorithms.

4.2.1 Before goal configuration

In the initial part no goal has been defined, so the execution time is split between the search of valid grasps, following one of the methods from Section 4.1, and the random growing of a forward tree *startTree* from the initial robot configuration. In Algorithm 1 (and in the experiments), the ratio between the two tasks has been empirically set to 80% and 20%, respectively. The initial exploration of the arm/-hand configuration space is useful for the second part of the planner, where a tree growing backwards from the goal has to connect to *startTree*.

Algorithm 1 Integrated grasp and motion planner.

```
1: procedure PLANNER                                          ▷ Planner main loop
2:     startTree ← c_init
3:     filterCapabilityMap()                                  ▷ If Cap. map is used for Alg. 2
4:     while numgoaltrees = 0 do                              ▷ PART 1: No goal yet
5:         p ← randomUniform(0, 1)
6:         if p < 0.2 then
7:             c_smp ← cspaceUniformSampling()
8:             extendTree(startTree)
9:         else
10:            c_goal ← findGoal()                            ▷ Algorithm 2
11:            if c_goal ≠ NULL then                          ▷ goal found
12:                new tree
13:                tree ← c_goal
14:                listGoalTrees ← updateGoalTrees(tree)
15:            end if
16:        end if
17:    end while
18:    p ← randomUniform(0, 1)                                ▷ PART 2: Goal found
19:    if p < 0.8 then
20:        indTree ← whichTreetoConnect()
21:        if connectTrees(startTree, listGoalTrees[indTree]) then
22:            return pathToGrasp
23:        end if
24:    else
25:        c_goal ← findGoal()
26:        if c_goal ≠ NULL then                              ▷ new goal found
27:            new tree
28:            tree ← c_goal
29:            listGoalTrees ← updateGoalTrees(tree)
30:        end if
31:    end if
32: end procedure
```


Let c denote a configuration in the C-space of the arm/hand system; c_{init} is the initial configuration, c_{smp} corresponds to a configuration obtained via forward kinematics (FK), and c_{goal} is a goal configuration. A goal hand pose is obtained via Algorithm 2 with one of the sampling approaches: PCA-based computation (Section 4.1.1) or capability-based computation (Section 4.1.2). After this, an IK solver is called to verify that the grasp pose is reachable and obtain the corresponding arm configuration.

Next, the validity of the hand pose must be evaluated to verify if there exist reachable ICRs. Reachable ICRs provide contact regions on the object surface, but do not guarantee that the hand configuration (finger positions) leading to them is free of self-collisions between the fingers. The transformation defined in Section 4.1.3 helps for this purpose because the fact of having the orientation of the hand with the palm parallel to the main axis of the object gives to each finger a greater space to be placed on without self-collision. Therefore, after the ICRs are computed a hand configuration for grasping the object must be found; this is iteratively done by exploring the potential FC grasps that the regions provide.

Algorithm 2 Find goal configurations.

```
1: procedure FINDGOAL()
2:     (x, y, z) ← sampleHandPosition()
3:     T_hand^object ← computeTransformation(x, y, z)
4:     c_goal ← IK(T_hand^object)
5:     if c_goal then
6:         if validGoal(T_hand^object) then                ▷ Algorithm 3
7:             return c_goal
8:         else
9:             return NULL
10:        end if
11:    else
12:        return NULL
13:    end if
14: end procedure
```

Name	Structure	Description
smp	$(SE(3) \cup \mathbb{R}^n)$	Sample in the Configuration Space
motion	*smp* parent *motion*	Sample and parents
$Tree$		Nearest Neighbours structure of motions
$goalTrees$	array$< Tree >$	Vector of trees grown from valid goal configurations
$ICRquality$	$\mathbb{N}$	Contains the value of the quality metric for every *smp*
$minICRquality$	$\mathbb{N}$	Minimum value of $ICRquality$ required for a *smp* to be considered a valid goal configuration
qualitymap	map$< Tree, ICRquality >$	Maps every $Tree \in goalTrees$ with its associated $ICRquality$
probabilities	array$< \mathbb{R} >$	*probabilities*[i] is the probability that $Tree$ in $goalTree$[i] will try to connect with $startTree$
regions	2-D array$< \mathbb{N} >>$	*regions*[i] contains all the index to the independent contact points of the ith finger

Table 4.1: Data types.

Method	Return	Description
randomUniform(a,b)	$\mathbb{R}$	Samples uniformly a real number in the range [a,b]. Set to [0,1] in the experiments
cspaceUniformSampling()	*smp*	Samples uniformly in the C-space
checkCollision(*smp*)	bool	Returns true if smp is not collision-free
extendTree($Tree$,*smp*)	$Tree$	Extend the Tree towards smp as used in the RRT algorithm
findValidGoal()	*smp*	Find a valid goal configuration
sampleGoalSpace(*smp*)	bool	Adaptive sampling in the goal space around the object
validGoal(*smp*)	bool	Given a *smp* computes the Independent Contact Regions reachable from that sample
numgoalTrees()	$\mathbb{N}$	Number of backward trees already created
whichtreetogrow()	$\mathbb{N}$	Selects a backward tree
updateTrees()		Updates $goalTrees$, *qualitymap* and *probabilities* structures
icr(T)	*regions*	Holds the computation of the Independent Contact Regions
filtercapabilitymap()	3-D array$< \mathbb{R} >$	Filters the points in the capability map
reachable(T)	bool	Consults the capability map for reachability

Table 4.2: Methods.

Algorithm 3 Verification of goal validity.

```
1: procedure validGoal(T_hand^object)
2:     regions ← ICR(T_hand^object)
3:     if regions.size() ≥ 2 then
4:         ICRquality ← computeICRqual()                    ▷ Eq.(1)
5:         if ICRquality ≥ minICRquality then
6:             findCollfreeHandConfig(T_hand^object, regions)
7:             return true
8:         else
9:             return false
10:        end if
11:    else
12:        return false
13:    end if
14: end procedure
```

4.2.2 Goal configuration found

When the first valid grasp configuration is found, the second part of the algorithm starts. During this phase, the algorithm tries to connect $startTree$ with one of the valid grasp configurations. For a valid goal (grasp) configuration, a new tree $goalTree$ starts to grow backwards. At this point, most of the efforts will be focused on connecting $startTree$ with $goalTree$. However, the search for new valid grasp configurations still continues, although with a lower priority. This continuous search of new configurations looks for better quality grasps or for goals that can be easier to connect to $startTree$.

The connection process is biased towards the grasps with higher Q_{ICR}. With n goal configurations found, each $goalTree[i]$ has an associated quality Q_{ICR_i}, and its probability to be connected is given by

$$probabilities_i = \frac{Q_{ICR_i}}{\sum_{j=1}^{n} Q_{ICR_j}} \tag{4.5}$$

The planning procedure comes to an end when the connection between the forward and one of the backwards growing trees is successful, and returns a path in the C-space that allows the robot to grasp the object.

Chapter 5

Implementation

During the project, different software packages and libraries have been used to help in the development of the integrated planning method. This chapter introduces the most important ones in order to give a better understanding of the development process of the thesis. It is important to notice that the capability map and the Independent Contact Regions were already available libraries at the DLR, although some improvements were required during the thesis and implemented inside its context.

5.1 Open Motion Planning Library

The open motion planning library (OMPL) is a library for sampling-based motion planning that contains implementations of many state-of-the-art planning algorithms [3]. The library allows the user to easily solve a variety of complex motion planning problems with minimal input. OMPL itself does not contain any other capability other than the pure planning and sampling algorithms, i.e., it does not contain any collision checking or visualization. Thus, OMPL is not tied to any fixed collision checker or visualization front end, giving the user the freedom to interface it with other software components. As a result, the user must select a computational representation for the robot and provide an explicit state validity/collision detection method. OMPL is entirely implemented in C++, and although it includes Python bindings they are not used in the context of this project. It is one of the

most important tools used in this work, since the core algorithm proposed has been developed as a class inherited from OMPL.

OMPL contains implementations of many sampling-based algorithms such as PRM, RRT, EST, SBL, KPIECE, SyCLOP, and several variants of these planners. The majority of the sampling-based motion planners require similar components to solve a planning problem: a sampler to compute valid configurations, a collision checker to quickly evaluate a specific robot configuration, and a local planner to connect two samples along a collision free path. OMPL provides most of these components, although some of them must be allocated or reimplemented by the user. The most important OMPL components are listed below.

One of the key classes is the *StateSpace*, which provides implementations for several common configuration spaces with different topologies, including R^n for Euclidean spaces and $SO(2)$ and $SO(3)$ for the space of rotations in 2D and 3D, respectively. In many cases the robot space can be defined by a combination of these basic spaces. Therefore, OMPL allows the definition of these kind of spaces through the *CompoundStateSpace* class, which allows state spaces to be formed from a combination of their subcomponents.

The *StateSampler* class implemented in OMPL provides methods for uniform and Gaussian sampling of the space.

The *StateValidityChecker* is used to evaluate if a single state collides with an environment obstacle and respects the constraints. A default checker is not provided by OMPL, as explained above. The user must provide a callback to this validity checker.

The *MotionValidator* class has the function of the local planner and checks to see if the whole motion of the robot between two states is valid. The standard implementation uses the interpolated movements between two states (computed by the *StateSpace*) to determine if a particular motion is valid. This is an approximate computation, as only a finite number of states along the motion are checked for validity (using the *StateValidityChecker*).

The *Planner* class is a base abstract class providing basic usage for planning, but most of its methods, like the method *solve* that actually solves the query, must be reimplemented by the different inherited planners to provide the full functionality. The planner described in Chapter 4 is defined as a reimplementation of the BiRRT planner included in the *RRTConnect*. One of the problems found in the reimplementation was the need to describe a planner that does not take any goal configuration as an input, which was not previously considered in OMPL.

Finally, *SimpleSetup* provides a way to encapsulate the various objects necessary to solve a geometric or control query in OMPL. When using *SimpleSetup*, the user only supplies all the needed objects for the planning solution.

Through the corresponding interfaces with The Kautham Project [1](version september 2013) and OpenRAVE [28], OMPL has been used in the thesis as the application containing the core code implementing the Integrated Grasp and Motion Planning.

5.2 The Kautham Project

The Kautham Project is a simulation tool developed at the Institute of Industrial and Control Engineering (IOC), Universitat Politècnica de Catalunya (UPC). The Kautham Project provides a tool for the development of robot motion planners and for telemanipulation using haptic devices.

The Kautham Project has been developed as a teaching and research tool in the field of robotics and motion planning. This framework provides to the user the complete functionality needed in the sample-based path planning process, such as the implementation of basic standard planners, the most used sampling policies, a visualization interface, and collision detector. The code has a modular structure that helps the user to add new functionalities without great effort. It has 4 main modules, which are called LIBPROBLEM, LIBGUI, LIBSAMPLING and LIBPLANNER.

LIBPROBLEM contains all the functionality related with the workspace and with the robot. It contains classes as the workspace, robot, link, etc. This library uses

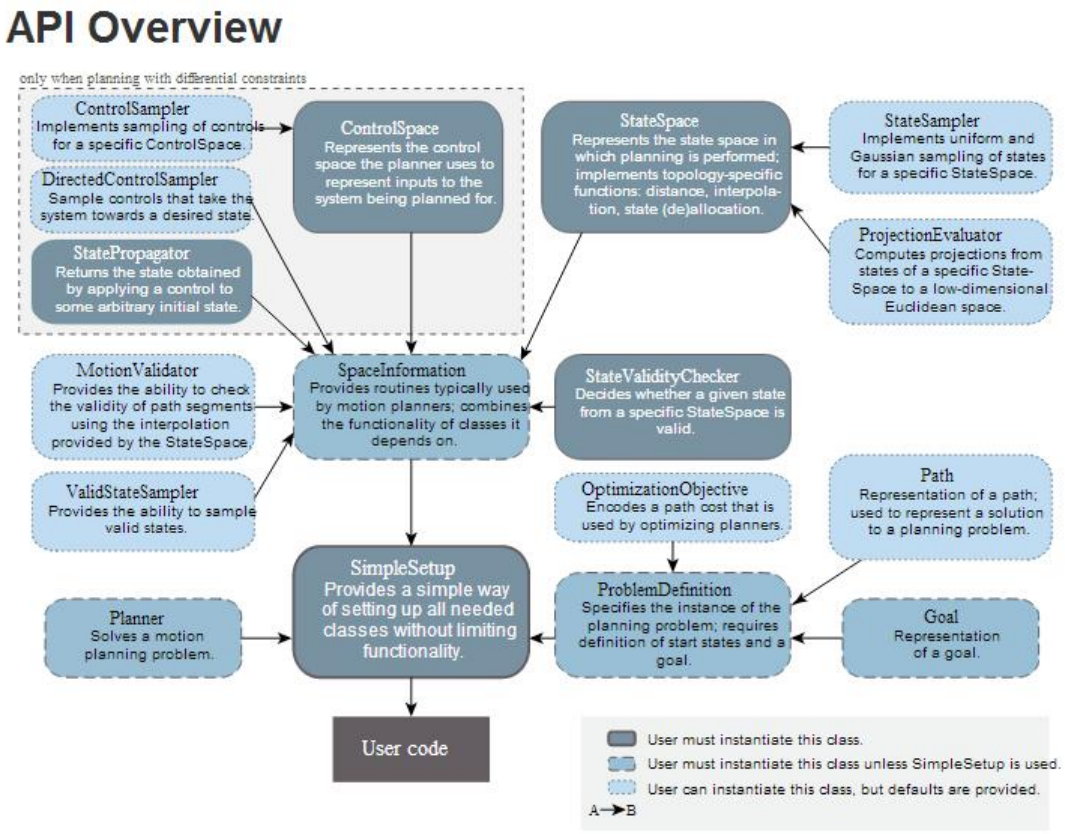

Figure 5.1: OMPL API Overview taken from OMPL website [3].

the PQP libraries for the collision queries. One of its most important characteristics is that it creates the environment and defines the planning problem to be solved by reading XML files using an XML parser library called PUGIXML. The XML file contains the definition of the robots, the position of every object in the environment and their mesh models. It also defines the main attributes needed for the planner query, such as the number of DOFs used by the planner, the start configuration, and some other parameters of the motion planners like the maximum length of a motion.

LIBGUI is developed using the Qt libraries and defines and provides the user interface. The standard user interface provided offers all the possibilities needed by any sampling-based planner. For example, it allows the user to define which configurations are the start and the goal of the query, but in the case of the algorithms implemented in this thesis, there were some other needs since no goal configuration

is preset. Therefore, a new GUI class was inherited from the existing one and it was slightly changed to be able to apply it to the selection of the object to be grasped (instead of selecting a goal sample).

LIBSAMPLING holds classes like *Sampler*, *Sample* and *SampleSet*. *Sampler* manages the filling of the sample sets and holds information about the connectivity between the samples. The *Sample* class is one of the main features of Kautham. In the case of Kautham the samples are initially taken in what is called the Control Space. The samples in the Control Space have a value between 0 and 1, which are the high and low limits of the index. The dimensionality of the Control Space is not necessarily the same as the C-Space dimensionality because Kautham is prepared to work with reduced dimensionality computed by the Principal Component Analysis. This allows to have different coupling between joints, which reduces the dimension of the problem, and can also be used to mimic human-like movements like the possible couplings present in an anthropomorphic hand.

LIBPLANNER contains the *Planner* base class, which is the parent class of all the other planners defined and used in Kautham. *Planner* owns objects of the *Sampler* class, *SampleSet* class and many others. *Planner* owns the method which actually solves the query. The planning algorithms used here were always the ones included by the developers. Recently, Kautham has integrated OMPL as a module, which allows the user to profit from the complete range of algorithms available in that library. This was achieved by building an interface wrapping the Kautham class definitions to make them usable by the OMPL planners.

5.3 Implementation of the Integrated Planner in The Kautham Project

As previously explained in 5.1, OMPL core does not provide any visual interface or collision checker, thus in this case they are provided by Kautham. In this integration process special attention was put in the conversion of the Samples in the Kautham control space to the States in the State Space used by OMPL. The interface to the

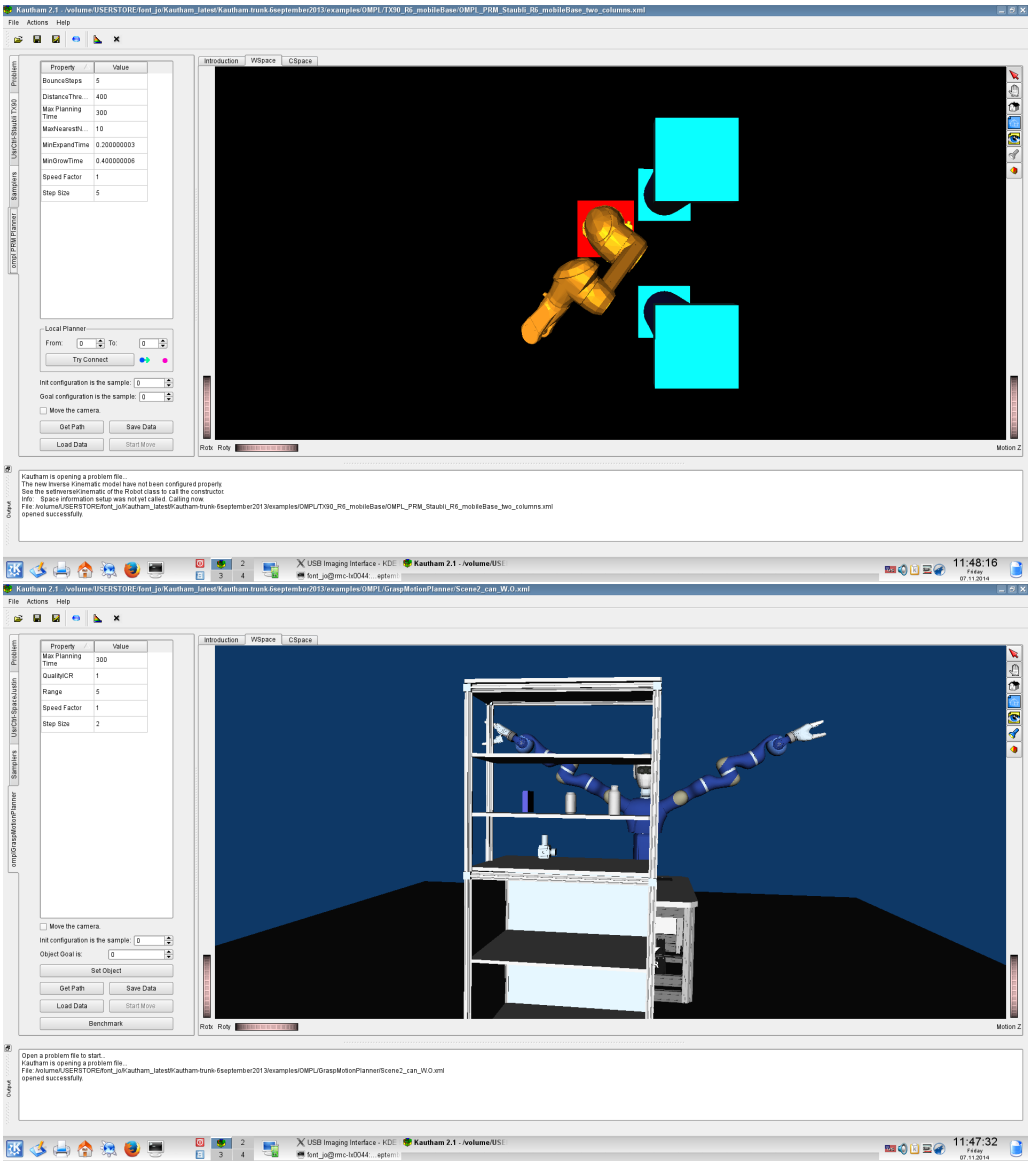

Figure 5.2: Different GUI implemented in Kautham.

planner is made through inherited planner classes called *ompl⟨plannertype⟩*, which own the actual OMPL planner. When these planners are called for solving a query, they internally call the OMPL planner solve method.

At the beginning the two described methods were implemented on Kautham using this OMPL interface. The general planner was written and derived from the OMPL module interfaced with Kautham in its LIBPLANNER module. Selecting which of the planners must be loaded through the XML files, different sampling strategies are called for grasp searching, allocating different kind of sampling subclasses from OMPL *ValidStateSampler*. The kinematics of the robot and the workspace definitions are the ones from Kautham. The process is similar as the one done for the implementation in OpenRAVE, and since that implementation is the definitive one that stays at DLR for future work, the process is explained in more detail later in sections 5.4 and 5.5.

5.4 OpenRAVE

OpenRAVE stands for Open Robotics Automation Virtual Environment. OpenRAVE provides also a framework for testing and developing motion planning algorithms in real-world robotics applications. Its main focus is the simulation and analysis of kinematic and geometric information related to motion planning. OpenRAVE was first developed by Rosen Diankov as his Ph.D. thesis in 2010 [28],and since then it has grown up to become one of the most used software packages by the robotics community. The proposed integrated planning method was also integrated to OpenRAVE in order to allow the future use of the module for future research purposes inside the DLR.

The framework was designed to have a plugin-based structure, allowing users and developers to extend and change the different plugins' functionality without the need of recompilation of the whole core library. The main core provides the Basic Interface Classes, which provide the basic classes for all the possible functionalities, but these are mainly abstract classes that must be completed by plugins. This core

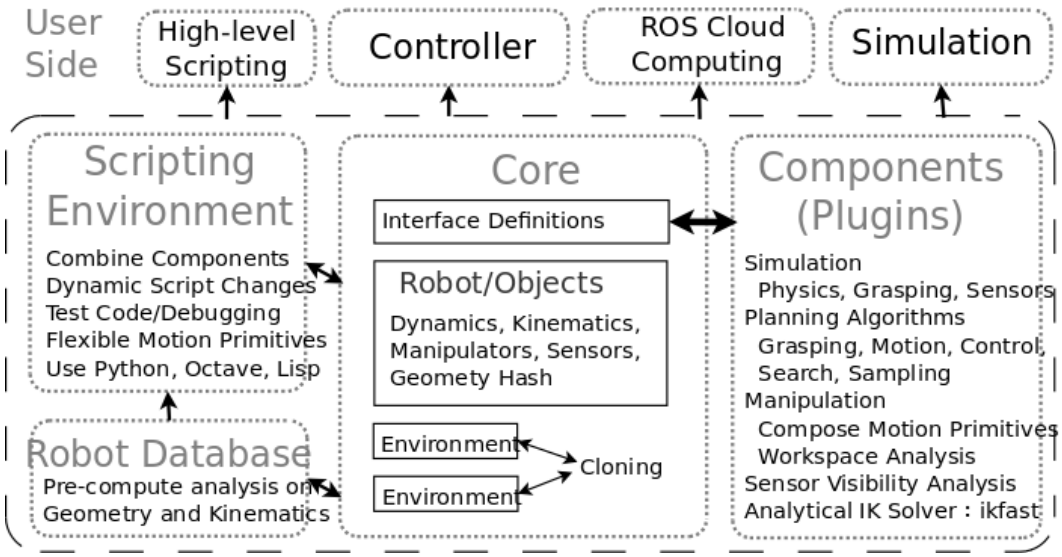

Figure 5.3: OpenRAVE architecture. (Taken from the website *http://www.openrave.org/*).

is written in C++ using the Boost C++ libraries, and its main purpose is holding the definition and functionality of the workspace, the environment and the robots, i.e., kinematics or geometrical information. On the other side, the plugins offer implementation of basic interfaces, and are loaded at run-time into the environment. To extend the capacities, plugins are linked to external libraries to have access to externally implemented classes and functions. OpenRAVE provides a main class called *EnviromentBase*, which gives access to all the available services. It supports the loading of robots and scenes, the communication and managing of plugins, the collision checking, etc. Another characteristic of OpenRAVE is the presence of a Python scripting layer, which gives access to the C++ functionality of the framework through a Python API. This eases the work with OpenRAVE and the definition of different planning problems.

5.5 Implementation of the Integrated Planner in OpenRAVE

As explained above, the method has been integrated in Kautham and in OpenRAVE, but as the second one is the one that stays at DLR to be used in the future, the explanation of the integration is based on this version.

The integrated planner was added to OpenRAVE using the *or_ompl* package. This plugin was developed by the Robotics Lab in the Robotics Institute at Carnegie Mellon University. It provides the interface to use OMPL inside the OpenRAVE plugin, thus giving the option to delegate planning requests to an OMPL planner.

The method is developed and coupled with *or_ompl* as a shared library to be used as an OpenRAVE plugin. The code of *or_ompl* was slightly modified to be able to work with planners as the one developed in this thesis, that does not take any goal configuration as an input.

The planner core is a planner class inherited from the OMPL *RRTConnect* class. This class is called *GraspMotionPlanner* and allocates instances to other main classes such as *HandArmStateSampler* or *SamplerCapability*. They are in charge of sampling hand-arm configurations inside the configuration space and of the sampling of potentially grasping positions in the workspace using the capability map, respectively. These classes inherit also from the classes of the ompl library *StateSampler* and *ValidStateSampler*. *HandArmStateSampler* was created with the only purpose of sampling the configuration space, taking into account that it is useless to sample during the complete process different finger configuration, thus increasing the performance of the probabilistic sampling method. Therefore, *HandArmStateSampler* only samples the arm space by keeping the finger joints at the starting value. On the other hand, *Samplercapability* is one of the most important parts of the approach. It owns instances of the capability map and has access to its arm workspace discretization and its querying methods. It also has an instance of the class interfacing to the library containing the ICR method, called *IndependentContactRegions*, which is totally independent from OMPL or OpenRAVE.

Another important class of the implementation is the *IvKinSpaceJustin* class, which is an abstract class to be used as a base for the right and left arms inverse kinematics solvers of Space Justin. These classes use an optimization-based solver for the inverse kinematic problem, and it was developed inside the Robotics Institute at DLR [29].

The planner is accessible through the Python script used for OpenRAVE. The only difference is that the user has to define OMPL as the planner to be used, and in the parameters it must be exactly specified which planner method is employed. For this planner there are also some extra parameters that can be defined by the user. Some of them are the limit of time given to the planner to find a solution, which thumb configuration has the hand mounted on the Space Justin arm, which arm is used for the planning task, or which object in the scene definition is the one to be grasped.

A typical example of a script calling the planner is as follows:

```
from openravepy import * # import all the functions from OpenRAVE Python API
env = Environment() # Create the environment
env.SetViewer('qtcoin') # Set the viewer
env.Load('path to scene')# Load the scene
robot = env.GetRobots()[0] # Choose the first robot in the scene
planner = RaveCreatePlanner(env, 'OMPL') # Create an OMPLPlanner
params = Planner.PlannerParameters() # Create parameters
params.SetExtraParameters(
""" <planner_type>GraspMotionPlanner< /planner_type>
<time_limit>15.0< /time_limit><object_goal>1< /object_goal>
<manip>1< /manip><hand>dlr_hit< /hand><handeness>right< /handeness>
""")# Set the extra parameters
planner.InitPlan(robot, params) # Initialize the planner
traj = RaveCreateTrajectory(env,"") # Create empty trajectory
result= planner.PlanPath(traj) # Plan and fill the trajectory
robot.GetController().SetPath(traj) # Run the trajectory
robot.WaitForController(0)
```

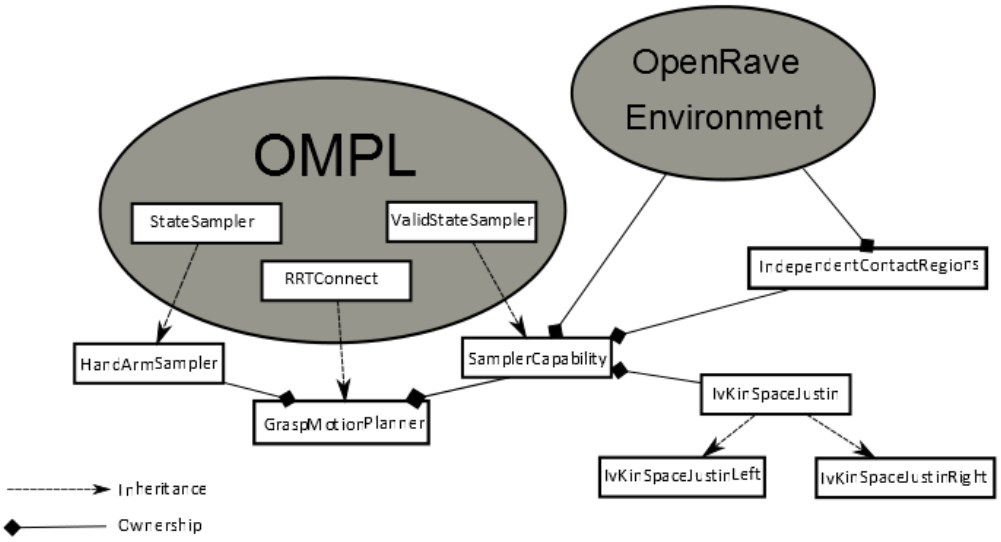

Figure 5.4: Library diagram.

As a summary, the method has been developed as a library for OpenRAVE. The user needs to define the scenario and the robots, these scenarios can be precomputed or acquired through appropriate sensors. The user must also decide which of the objects is the one to be grasped, and with which arm. Optionally the user can define an initial configuration. If this initial configuration is not previously set, the planner takes the actual configuration of the robot in the scene as the starting one.

The documentation of the library is presented together with this descriptive document. It also shows the general inheritance structure explained in this section.

Chapter 6

Results and discussion

The proposed approach has been implemented as a specialized RRT planner inside the path planning frameworks OpenRAVE [28] and *The Kautham Project* [1]. The RRT planner is based on the implementation of RRT-Connect from the Open Motion Planning Library (OMPL) [3]. The computation of the reachable ICRs follows the algorithm proposed in [2], which uses a modified Voxmap-Pointshell (VPS) algorithm for an efficient detection of hand-object collisions [30].

The application examples are solved for SpaceJustin, the upper body of a humanoid robot described in Section 3.1. To evaluate the performance of the integrated planner, two different environments have been tested, one tabletop and one cupboard scenario, grasping different objects with and without obstacles in the way (Fig. 6.1). Objects different than the target are considered as obstacles.

The same test scenarios were replicated inside the planning framework *Simox* [31], which includes the planning approach described in [20], hereafter referred to as Grasp-RRT (following the name given by their authors). Thus, a fair comparison of the approaches can be obtained, using the same platform and same geometric models for collision detection. The approaches proposed in this thesis will be referred to as PCA-RRT and CAP-RRT, depending on whether the planner searches for grasp poses using the PCA technique or the capability map, respectively.

The comparison is performed for two examples grasping two objects, a soda can and

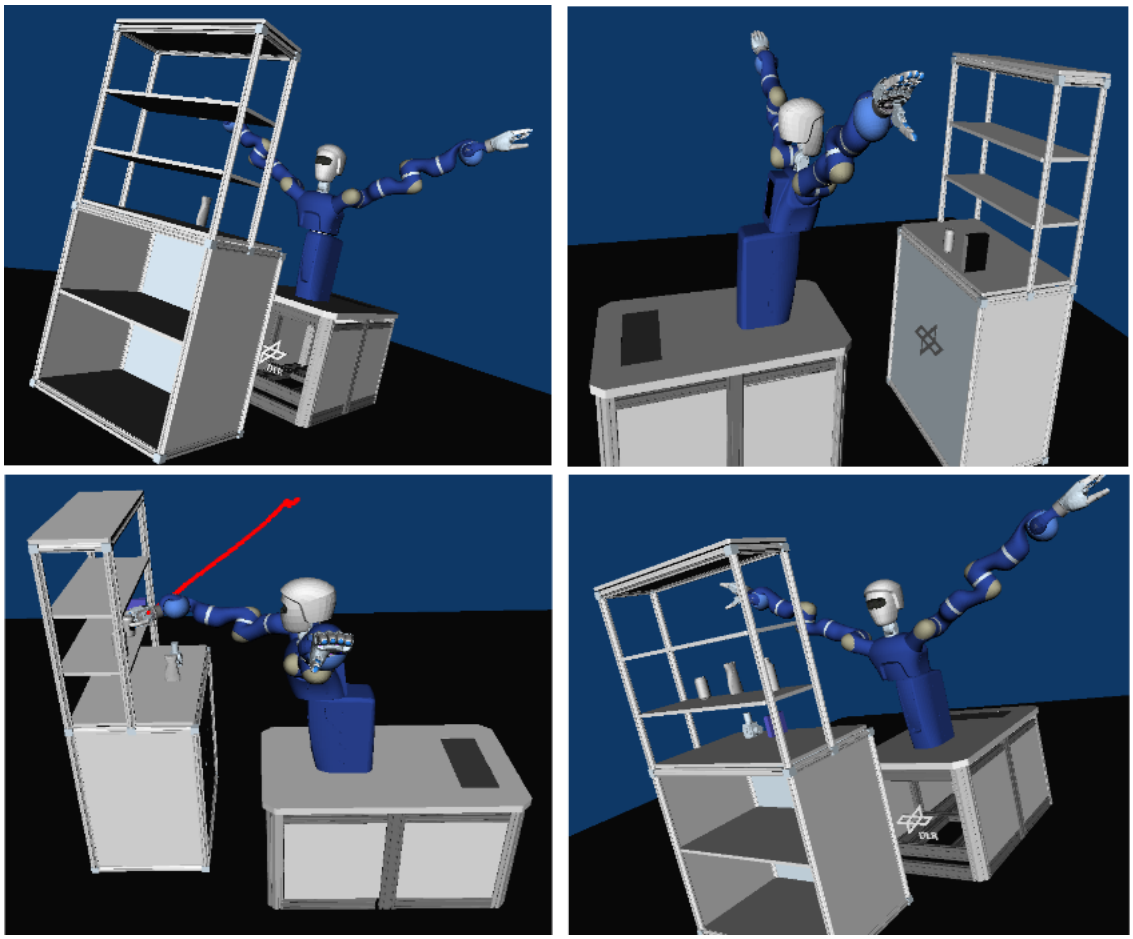

Figure 6.1: Tabletop and cupboard scenarios, with and without obstacles, for testing the planning approaches.

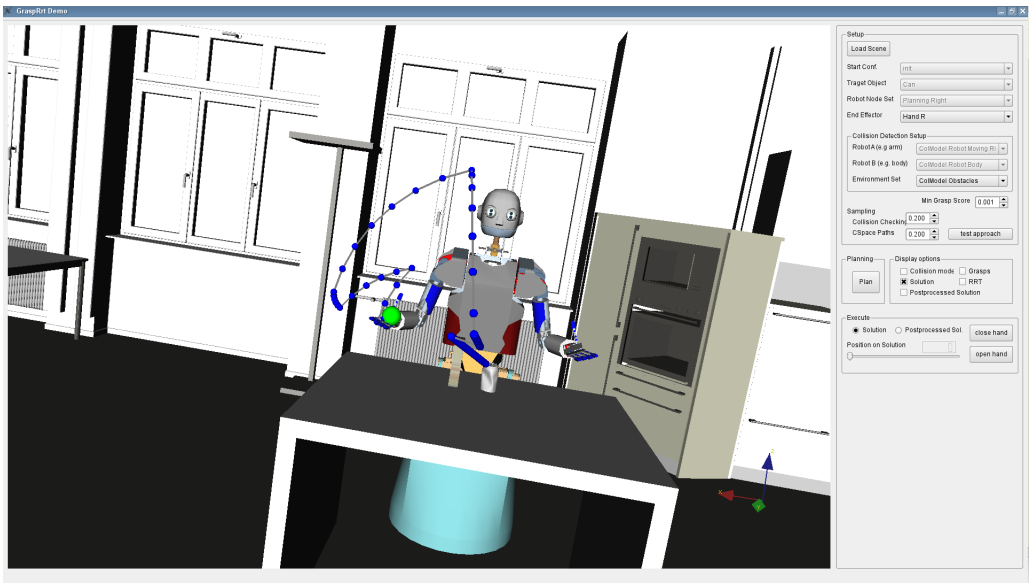

Figure 6.2: Grasp-RRT in Simox (this scenario was not used in the comparison).

Table 6.1: Time to grasp the soda can (s).

Scenario		Grasp-RRT	PCA-RRT	CAP-RRT
Tabletop	N.O	15.32 ± 18.33	13.20 ± 9.68	5.58 ± 3.30
	W.O	18.86 ± 20.87	28.43 ± 17.57	12.66 ± 8.93
Cupboard	N.O	30.23 ± 25.96	15.18 ± 11.08	8.95 ± 7.49
	W.O	38.88 ± 34.83	26.34 ± 16.14	23.79 ± 20.00

a shampoo bottle. Tables 6.1 to 6.4 summarize the computational times and success rate for the planning approaches inside *Simox*. N.O. and W.O. stand for scenarios with no obstacles or with obstacles, respectively. 50 runs have been executed for every test, allowing them to run for a maximum time of 100 seconds.

The results show that the integration of motion and grasp planning procedures successfully leads to finding a feasible grasp configuration with its corresponding path for the arm/hand motion, as demonstrated also in the video attachment. Note that the Grasp-RRT is a method that works exclusively based on FK computations, while the two methods proposed in this document - based on ICRs computation - need some IK calls: to guarantee that the potential grasps are in fact feasible for the arm (PCA-RRT), or to obtain an arm configuration that leads to a given reachable

Table 6.2: Success rate for grasping the soda can (%).

Scenario		Grasp-RRT	PCA-RRT	CAP-RRT
Tabletop	N.O	100 %	100 %	100 %
	W.O	100 %	100 %	100 %
Cupboard	N.O	98 %	98 %	100 %
	W.O	86 %	96 %	98 %

Table 6.3: Time to grasp the shampoo bottle (s).

Scenario		Grasp-RRT	PCA-RRT	CAP-RRT
Tabletop	N.O	32.99 ± 28.78	11.59 ± 8.09	6.37 ± 3.23
	W.O	36.3± 33.29	24.88 ± 20.80	10.93 ± 6.06
Cupboard	N.O	36.3 ± 33.29	23.59 ± 22.20	5.59 ± 2.83
	W.O	37.95 ± 31.08	34.3 ± 19.63	11.09 ± 6.10

Table 6.4: Success rate for grasping the shampoo bottle (%).

Scenario		Grasp-RRT	PCA-RRT	CAP-RRT
Tabletop	N.O	94 %	100 %	100 %
	W.O	86 %	94 %	100 %
Cupboard	N.O	86 %	98 %	100 %
	W.O	90 %	96 %	100 %

Table 6.5: Time distribution for the CAP-RRT planning approach.

Total	Collision	Reachability	ICR	Rest
12.15 s	10.00 s	0.54 s	0.16 s	1.46 s
100 %	82.3 %	4.4%	1.32 %	12.02 %

hand pose (CAP-RRT).

Despite working with different philosophies, the Grasp-RRT and the PCA-RRT approaches have a comparable performance in several cases, although the PCA-RRT seems to have less variability in the computational times and behaves better in most of the tested scenarios. However, the best approach turns out to be the CAP-RRT, i.e. the planner that uses information from the capability map to restrict the directions that the robot should use to try and grasp the object. It is faster that the Grasp-RRT (1.5 to 6.5 times, depending on the scenario) and has less variability in the time required to get a solution.

To gain some insight into the time distribution of the planning approach, the averaged times for another 50 runs of CAP-RRT in the tabletop scenario with obstacles are presented in Table 6.5. The times are analyzed for three critical parts: collision detection, computation of reachable points for the hand, computation of reachable ICRs, and for the remaining parts (RRT generation and connectivity). The results show that most of the time spent in the complete procedure goes into collision detection, which is performed using the original meshes of the robot model. However, using simplified models can reduce the weight of the collision detection in the total planning time.

To evaluate the improvement obtained by using simpler collision models, the example of grasping the soda-can on the cupboard was run 50 times with two different collision models, the original mesh and a simplified version. These experiments where performed in the OpenRAVE implementation, due to the limitations of the Kautham implementation used (september 2013) to arbitrarily set which meshes are selected to check for collision. The different meshes were built by reducing the number of the triangles using Blender [32], an open-source graphical modeling soft-

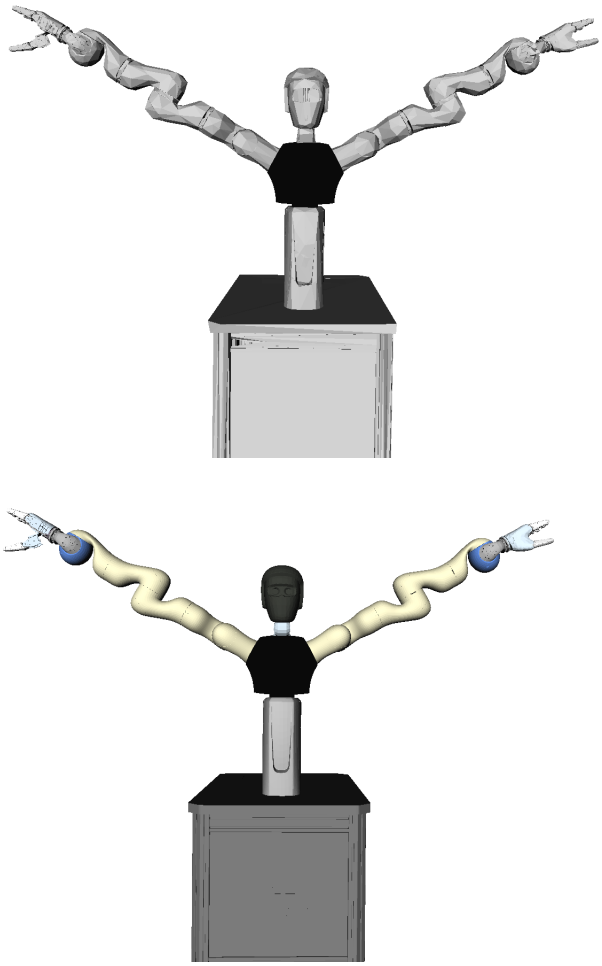

Figure 6.3: Collision models for SpaceJustin: simple model (up) and full model (down).

ware. An optimal simplification would use the approximation of every link by a convenient geometrical primitive. The results confirm that collision detection time is a key factor in the performance of the planning algorithm.

Another potential gain in time can be achieved by improving the method for selecting the final finger poses for grasping the object given the ICRs, so that configurations with self-collisions between the fingers can be efficiently avoided.

The approach which turned to have the best performance, the so called CAP-RRT, has been tested on the real robot to prove the physical validity of the method. For

Table 6.6: Collision detection: Time comparation (s).

Scenario	Simple Model	Full Model
Cupboard	5.37 ± 2.89	6.73 ± 3.13

Figure 6.4: Diagram of implementation for the real experiment.

this real test two different objects where placed on a table in the DLR telemanipulation laboratory. A simple OpenRAVE environment was built having the models of the objects and a similar model to the table (in this case no big obstacles were used). The implementation runs as specified in Figure 6.4. A visual module obtains the position and orientation with respect to the robot base of a predefined object, which is fed into the planner software module that starts the computation of the path. Once the path is obtained, the corresponding path steps are sent to the controllers of the robot as joint angles inputs. A last step to close the fingers applying pressure and properly grasp the object was performed. For this step, the position and orientation of the fingers on the surface of the object where obtained. With this information, the fingers are switched to be controlled by a Cartesian position controller, and the fingers were commanded to push in the negative normal direction of the object normal at that contact point until the torque sensor of the fingers reaches a predefined threshold value. After the complete path is executed, the robot is commanded to a rest pose by a joint interpolation method to prove that the object is actually properly grasped and does not fall from the robot hand.

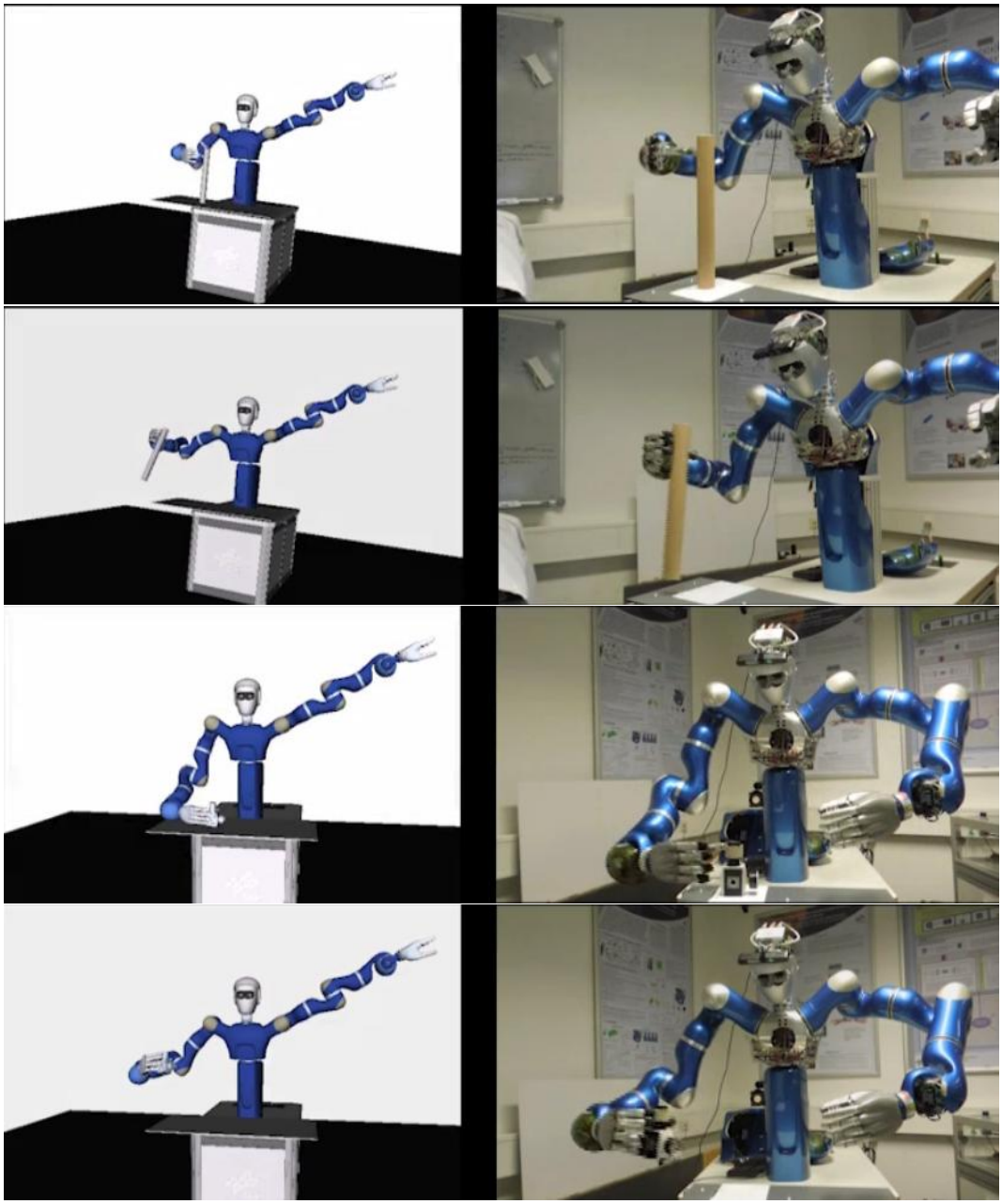

Figure 6.5: Snapshots of the real experiment with SpaceJustin.

Chapter 7

Environmental Analysis

Every project requires an analysis of is its environmental impact. The impact may be done during the developing process or by the time of the use and execution of the work done.

7.1 Environmental Impact

Given the nature of this project, mainly a software tool to be applied on robotics systems, the impact on the environment is almost zero and could be just defined with the considerations of the power spent by the computers during the development. No special wastes or emissions are generated at any point of the process.

7.2 Socio economic Impact

The impact of the project has to be also analyzed from a social and economical point of view. In this case, the work is done inside a robotics research center and with pure research purposes, which makes it difficult to recognize in it a direct impact in the industry and in the economy such as the creation or destruction of jobs. The research results of the project can help to go a step further in the development of robotic technologies, which can mean a real influence in the future in the modernization

of the industry towards process automation. This indirectly involves the change of less qualified jobs for more qualified ones in the programming and operating of the robotic systems. It also should make a difference in the assistance of handicapped persons needing help in their daily life.

Chapter 8

Costs Analysis

For the economical study of the project, two stages must be considered: the development costs and the costs of execution. These two costs form the total cost of the project for those who do not own the needed hardware specified in the execution costs. For the persons that already own that part, the only direct costs are the ones from the developing part.

8.1 Development Costs

The development of the project has only the costs associated to the work of an engineer and programmer, checking the hours dedicated to the study and evaluation of the project and the lines of code. The lines of code are counted with a Linux terminal application which counts the lines of code present in a project, eliminating the blank lines. Since the author is not an expert programmer, the estimated number of lines of code that he can produce in a day is 150 lines, and assuming a working cost of 150 € per day, the cost of a line of code is $1€/line$.

Table 8.1: Costs of code.

	Number of lines	Cost
Header files .h	1874	1874 €
Source files .cpp	6820	6820 €
Total	8694	8694 €

The total programming cost must be added to the cost in terms of time of the previous study and development of the project. 50 hours of an engineer were dedicated to the study and previous work before starting the software implementation with a unitary price of 30€/*hour*. The total developing costs raise up to 10194 €.

Table 8.2: Total developing cost.

Concept	Total cost
Study	1500 €
Programming	8694 €
Total	10194 €

8.2 Execution Costs

The execution costs must be taken into account for those who do not have a robotic arm and an anthropomorphic hand. For this project, all the tests where done with SpaceJustin, but an LBR robotic arm with a DLR-HIT HAND mounted on top would be enough. Although other options are possible, the cost analysis is done for these components. The sum of these costs adds up to 115500 €.

Table 8.3: Execution costs.

Component	Cost
LBR	74500 €
DLR HIT HAND	45000 €
Total	115500 €

Then, the total cost of the project is 10194 € for the development costs plus the 115500 € needed to obtain all the robotic elements.

Conclusions

The project has successfully reached the goals, and the implementation meets all the requirements posed in the problem definition. A complete grasp and path planner was developed, which includes the search of grasps inside the main loop and does not have a pre-defined configuration as a goal. By using tools available at the DLR such as the Independent Contact Regions implementation and the Capability map concepts, the planner successfully finds a grasp and an arm path, and the grasps are robust against the fingertip positioning errors.

The performance of the planner has been proven to be better than other existing integrated planners following the same idea. Although two approaches were tried during the development of this work, one of them had a much better performance and was finally implemented, and is now available inside the Robotics institute at DLR to be used as an integrated motion and grasp planner for future applications. It is also meant to be used as a planning tool inside a higher abstraction task planning framework that plans the order and sequence of steps to be applied in order to perform a task.

On a final note, the software implementation just relied on open source software or on software owned by the institution where the project was developed.

Future work

This project proposed algorithms and methods to find valid motions towards a good grasp. Some improvements could be applied in the future: some of them are meant to improve the performance of the method, thus making it more efficient, while the others can be thought as a way to extend the functionality towards more complete and robust functions in the robotics manipulation community.

The first group includes all the possible improvements involving any of the basic technologies and algorithms present in the global method, which would automatically become an advantage for this work. But also there are some considerations independent from the basics algorithms. As mentioned in this document, a critical issue in finding feasible grasps is the presence of a lot of self-collisions between the fingers. To minimize these effects, two possible solutions could be considered in the future. First, a better algorithm or heuristic to decide which of the colliding points on the object for every finger are selected to start the growing process of the Independent Contact Regions. Another option to tackle this problem is the redefinition of the workspace of the fingers. During the complete process, the workspaces of every finger were the same. One possible alternative is to reduce the dimensions of each finger's workspace depending on its position on the hand, for instance the index finger on the right hand should have all the freedom to move to the left, but it should have limited movement to the right due to the presence of the other fingers. This would limit the sizes of the independent regions, but would increase the chances to directly find a valid grasp without self-collisions.

On the other hand, it would be interesting to extend the functionality of the planner

and adapt it to be used in bi-manual manipulation. To grasp big objects often one hand is not enough, but the same integrated planning principle for finding all the possible regions on the object considering the actions of two hands could be applied. Another functionality was already implemented in a simple way. The goal was to use the implemented planner inside a pick and place task, where the object has to be picked with one hand and placed with the other. The primitive implementation just considered long cylindrical objects like a tube where two hands could easily grasp together the object. At the beginning, the planner runs as in single manual grasping with the only difference that the grasp must be validated in a second phase. The extra validation applied in this case is the existence of a valid FC grasp with the other hand and with the object virtually placed on the goal position, on the part of the object which is not covered by the first one. When the grasps are validated in the initial and goal position and the plan towards the first grasp is solved, a position between the two arms is sampled to find the configurations where it is feasible to exchange the object from one hand to the other. Once this is found, some motion planners act to find the complete sequence of movements leading to the complete pick and place sequence. A diagram presenting the sequence of actions is showed in 8.1.

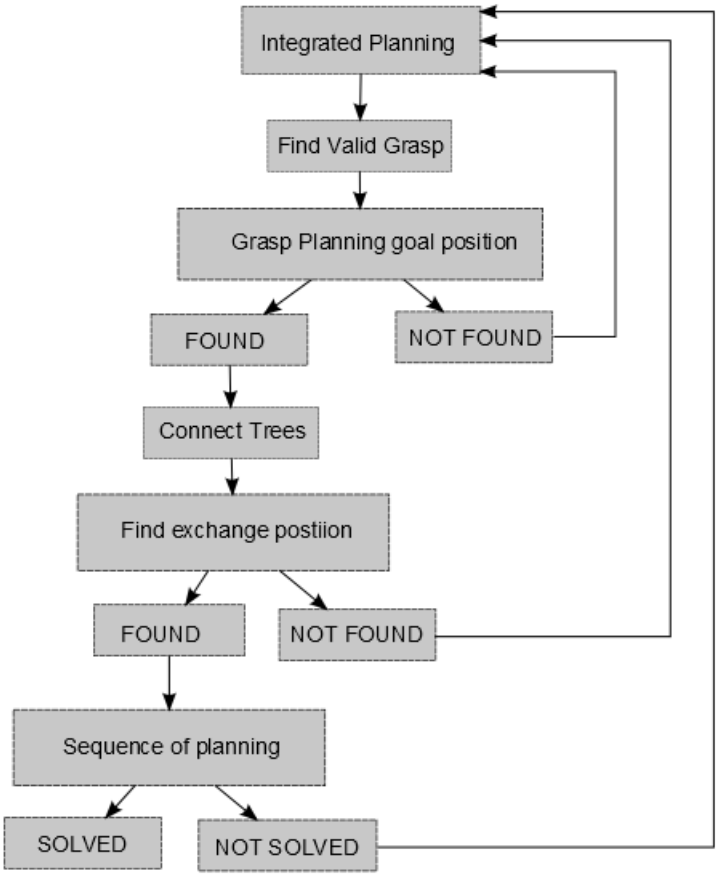

Figure 8.1: Diagram explaining the process towards a bi-manual pick and place application.

Acknowledgement

I want to dedicate some lines to say thank you to all the people who meant a big help during all this time.

First I have to thank Dr. Jan Rosell and Dr. Máximo Roa, the directors of the thesis, for the great support they gave me at all times, and for being always there to solve any doubt that could appear.

I also want to thank everyone at the Robotics and Mechatronics Center at the DLR for the amazing time I was able to spent there, and specially to the "Robotic Grasping and Manipulation Team" since without their support this work would have been impossible.

And lastly, I have to thank my family and friends for all the support that they always provided to me and their visits to Germany that gave me the strength to complete the job.

Bibliography

[1] J. Rosell, A. Pérez, A. Aliakbar, Muhayyuddin, L. Palomo, and N. García. The Kautham Project: A teaching and research tool for robot motion planning. In *Proc. IEEE Int. Conf. on Emerging Technologies and Factory Automation, ETFA*, 2014. 5, 16, 51, 61

[2] B. A. Dang-Vu, M. A. Roa, and C. Borst. Extended independent contact regions for grasping applications. In *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems - IROS*, pages 3527–3534, 2013. 5, 23, 61

[3] I.A. Şucan, M. Moll, and E. Kavraki. The open motion planning library. *IEEE Robotics and Automation Magazine*, 19(4):72–82, 2012. 6, 49, 52, 61

[4] K. Hertkorn, M. A. Roa, M. Brucker, P. Kremer, and C. Borst. Virtual reality support for teleoperation using online grasp planning. In *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems - IROS*, pages 2074–2074, 2013. 13

[5] K. Hertkorn, B. Weber, P. Kremer, M. A. Roa, and C. Borst. Assistance for telepresence using online grasp planning. In *Proc. IEEE-RAS Int. Conf. on Humanoid Robots*, pages 507–513, 2013. 13

[6] O. Porges, T. Stouraitis, C. Borst, and M. A. Roa. Reachability and capability analysis for manipulation tasks. In M. Armada, A. Sanfeliu, and M. Ferre, editors, *ROBOT2013: First Iberian Robotics Conference*, Advances in Intelligent Systems and Computing 253, pages 703–718. Springer, 2014. 13, 29

[7] S. M. LaValle. *Planning Algorithms*. Cambridge University Press, Cambridge, U.K., 2006. Available at http://planning.cs.uiuc.edu/. 15, 16

[8] S.R Buss. Introduction to inverse kinematics with jacobian transpose, pseudoinverse and damped least squares methods. *IEEE Journal of Robotics and Automation*, 17:1–19, 2004. 16

[9] J. Kuffner and S. M. Lavalle. RRT-connect: An efficient approach to single-query path planning. In *Proc. IEEE Int. Conf. Robotics and Automation - ICRA*, pages 995–1001, 2000. 17

[10] M. Zucker, N. Ratliff, A. D. Dragan, M. Pivtoraiko, M. Klingensmith, C. M. Dellin, J. A. Bagnell, and S. S. Srinivasa. CHOMP: Covariant hamiltonian optimization for motion planning. *Int. J. Robotics Research*, 32(9-10):1164–1193, 2013. 17

[11] S. Karaman and E. Frazzoli. Sampling-based algorithms for optimal motion planning. *Int. J. Robotics Research*, 30(7):846–894, 2011. 17

[12] J. Bohg, A. Morales, T. Asfour, and D. Kragic. Data-driven grasp synthesis - a survey. *IEEE Trans. Robotics*, 30(2):289–309, 2014. 18

[13] M. A. Roa and R. Suárez. Grasp quality measures: Review and performance. *Autnonomous Robots*, 2014. DOI: 10.1007/s10514-014-9402-3. 18

[14] C. Ferrari and J. Canny. Planning optimal grasps. In *Proc. IEEE Int. Conf. Robotics and Automation - ICRA*, pages 2290–2295, 1992. 18, 23

[15] M. A. Roa and R. Suárez. Computation of independent contact regions for grasping 3D objects. *IEEE Trans. Robotics*, 25(4):839–850, 2009. 18

[16] D. Berenson, R. Diankov, K. Nishikawi, S. Kagami, and J. Kuffner. Grasp planning in complex scenes. In *Proc. IEEE-RAS Int. Conf. on Humanoid Robots*, pages 42–48, 2007. 19

[17] D. Berenson and S. Srinivasa. Grasp synthesis in cluttered environments for dexterous hands. In *Proc. IEEE-RAS Int. Conf. on Humanoid Robots*, pages 189–196, 2008. 19

[18] D. Berenson, S. Srinivasa, D. Ferguson, A. Collet, and J.J. Kuffner. Manipulation planning with workspace goal regions. In *Proc. IEEE Int. Conf. Robotics and Automation - ICRA*, pages 618–624, 2009. 19

[19] D. Berenson, S. Srinivasa, and J. Kuffner. Task space regions: A framework for pose-constrained manipulation planning. *Int. J. Robotics Research*, 30(12):1435–1460, 2011. 19

[20] N. Vahrenkamp, T. Asfour, and R. Dillmann. Simultaneous grasp and motion planning. In *IEEE Robotics and Automation Magazine*, pages 43–57, 2012. 19, 61

[21] M.A Roa, C. Zhaopeng, I.C Staal, J.N Muirhead, A. Maier, B. Pleintinger, C. Borst, and N.Y Lii. Towards a functional evaluation of manipulation performance in dexterous robotic hand design. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 6800–6807. IEEE, 2014. 22

[22] M.A. Roa and R. Suárez. Finding locally optimum force-closure grasps. *Robotics and Computer-Integrated Manufacturing*, 25(3):536–544, 2009. 24

[23] M. A. Roa, K. Hertkorn, C. Borst, and G. Hirzinger. Reachable independent contact regions for precision grasps. In *Proc. IEEE Int. Conf. Robotics and Automation - ICRA*, pages 5337–5343, 2011. 25

[24] F. Zacharias, C. Borst, and G. Hirzinger. Capturing robot workspace structure: Representing robot capabilities. In *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems - IROS*, pages 3229–3236, 2007. 28

[25] J. Rosell, R.Suárez, and A. Pérez. Path planning for grasping operations using an adaptive PCA-based sampling method. *Autonomous Robots*, 35(1):27–36, 2013. 33, 37

[26] J. Rosell, R. Suárez, A. Pérez, and C. Rosales. Including virtual constraints in motion planning for anthropomorphic hands. In *Proc. IEEE Int. Symp. Assembly and Manufacturing - ISAM*, pages 1–6, 2011. 35, 40

[27] R. Balasubramanian, L. Xu, P.D. Brook, J.R. Smith, and Y Matsuoka. Human-guided grasp measures improve grasp robustness on physical robot. In *Proc. IEEE Int. Conf. Robotics and Automation - ICRA*, pages 2294–2301, 2010. 35

[28] R. Diankov. *Automated Construction of Robotic Manipulation Programs*. PhD thesis, Carnegie Mellon University, Robotics Institute, August 2010. 51, 55, 61

[29] R. Konietschke and G. Hirzinger. New inverse kinematics algorithms combining closed form solutions with nonlinear optimization for highly redundant robotic systems. 58

[30] M. Sagardia, T. Hulin, C. Preusche, and G. Hirzinger. Improvements of the voxmap-pointshell algorithm - fast generation of haptic data structures. In *Proc. 53rd Int. Wissenschaftliches Kolloquium*, 2008. 26, 61

[31] N. Vahrenkamp, M. Kröhnert, S. Ulbrich, T. Asfour, G. Metta, R. Dillmann, and G. Sandini. Simox: A robotics toolbox for simulation, motion and grasp planning. In *Intelligent Autonomous Systems 12*, pages 585–594. Springer, 2013. 61

[32] Blender Online Community. *Blender - a 3D modelling and rendering package*. Blender Foundation, Blender Institute, Amsterdam, 65